Document Title: IoT/Time series data and real-time streamings with NiFi
Date: September 2019
Version: 8.0
Notes: pre-created OneNode-cluster
Author: Filippo Lambiente, André Molenaar

IoT/Time series data and real-time streaming with NiFi

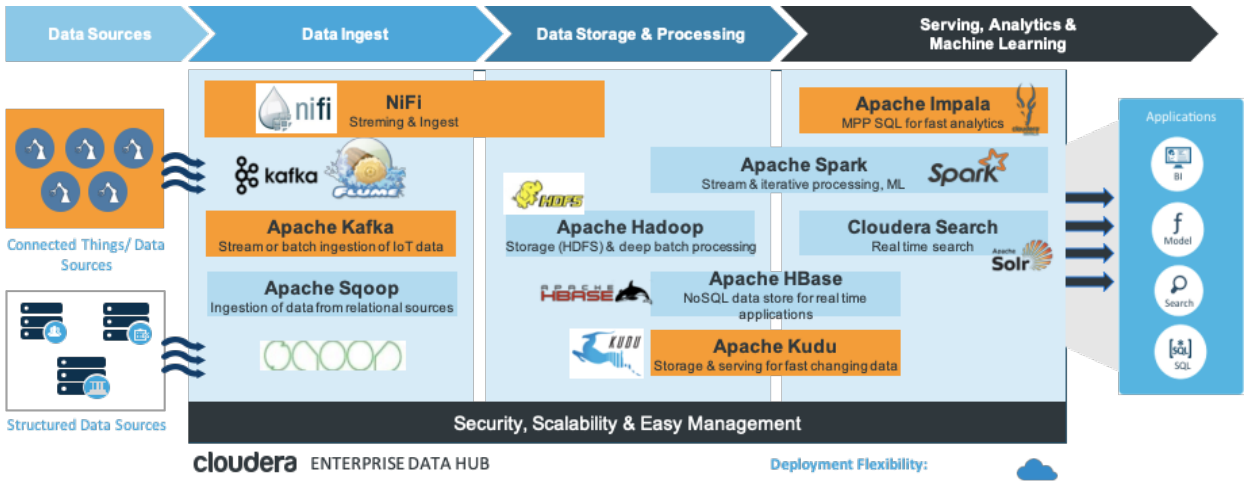
Table of Contents

Objectives

The goal of this track is to drive you through the implementation of an end-to-end scenario for real-time, time series, data ingestion with the Cloudera platform.
In particular:

- to create a simple & quick end-to-end demo in the IoT context from Data Ingest, to Data Storage, Processing and Serving;
- Eliminate the need of writing code
- leveraging the components of the platform and
- keeping the moving parts to a minimum

The picture below depicts the architecture of this demo scenario:



We highlight in orange the components we're going to use

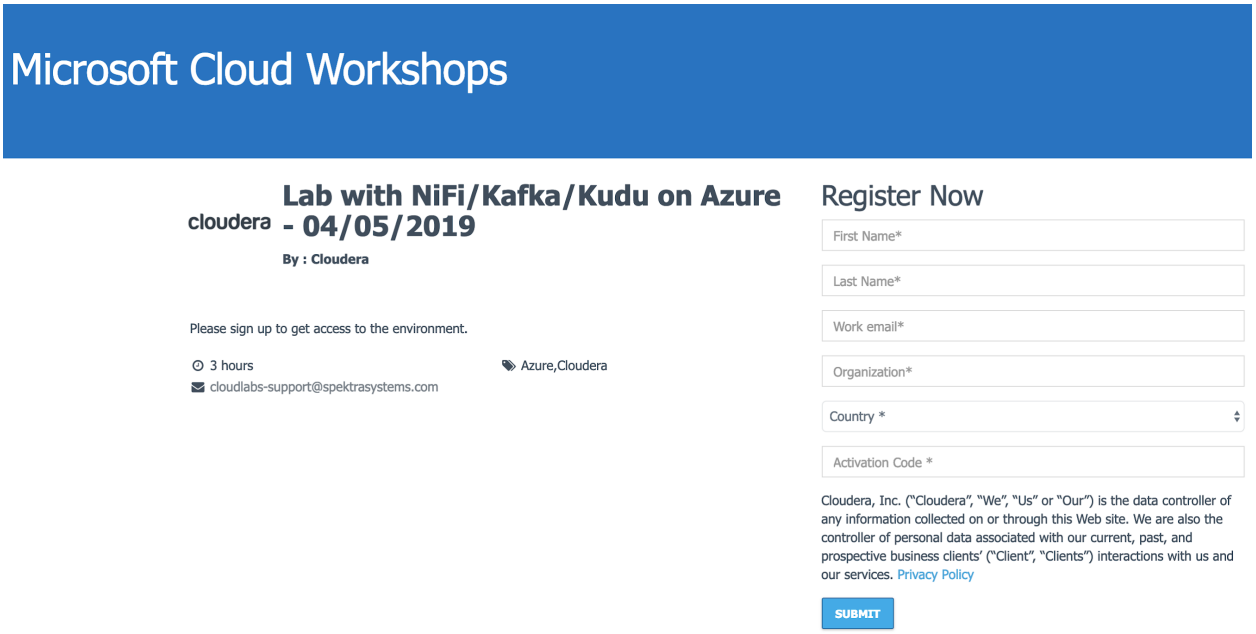
Labs that we will go through

Here is the list of the labs that will need to do in order to achieve our goals:

1. Get a pre-provisioned Cloudera cluster from Azure Marketplace
2. Preparing the cluster for real-time streaming ingestion
3. Get the Connected Things/Data Sources
4. Define the Kudu tables for the Connected Thing time series data
5. Create the real-time pipeline that reads data from the Connected Things and insert them into Kudu
6. Run and verify the real-time ingest pipeline
7. Improve Kudu table partitioning strategies and adjust the ingest pipeline
8. Versioning
9. Closing discussion

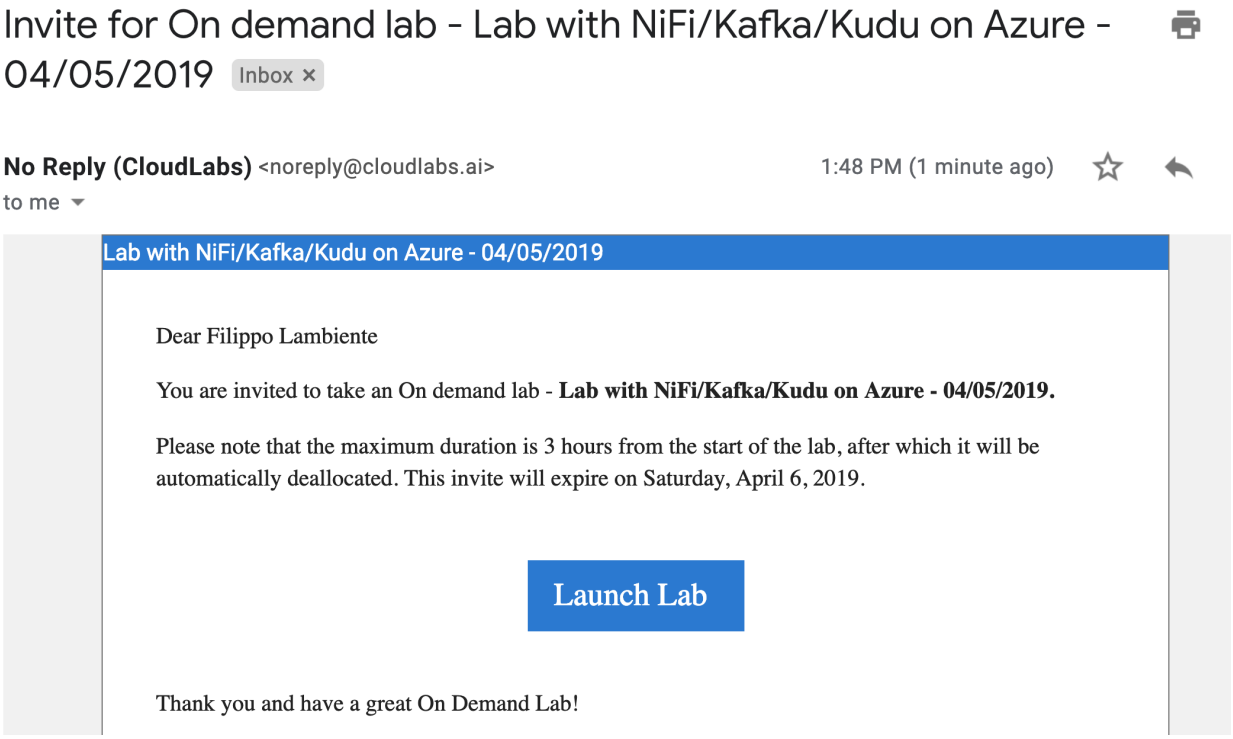
Lab 1 – User Registration for Cloudera on Azure hands on Labs

Go to the following link http://bit.ly/2Ufc6JN for the registration page

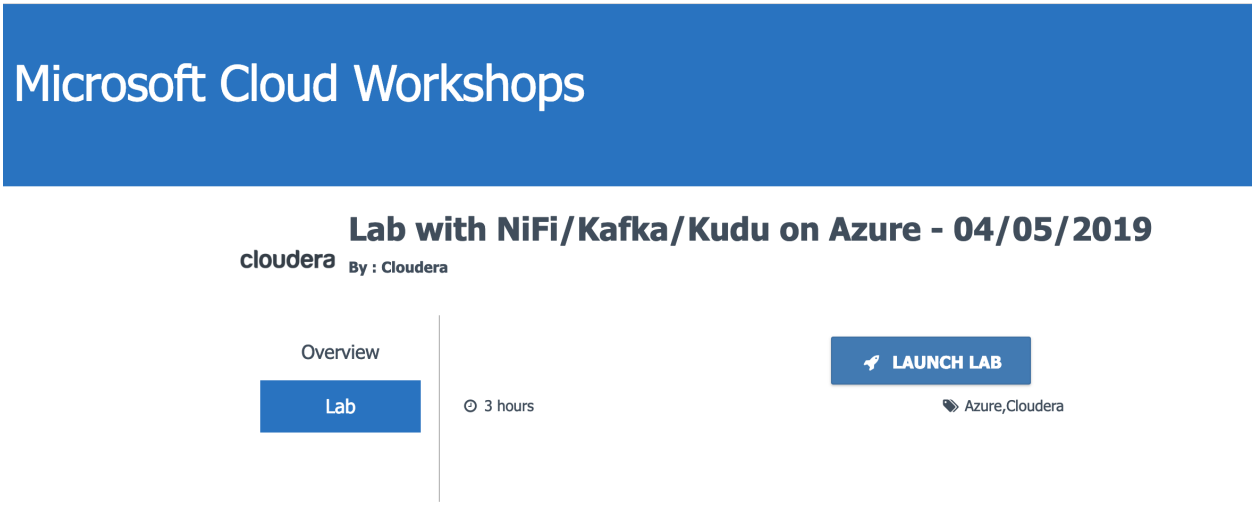


Fill in the fields and click **SUBMIT**
Your instructor will give you the **Activation Code**
Only corporate emails can be used to register (no gmail, yahoo, outlook or hotmail)

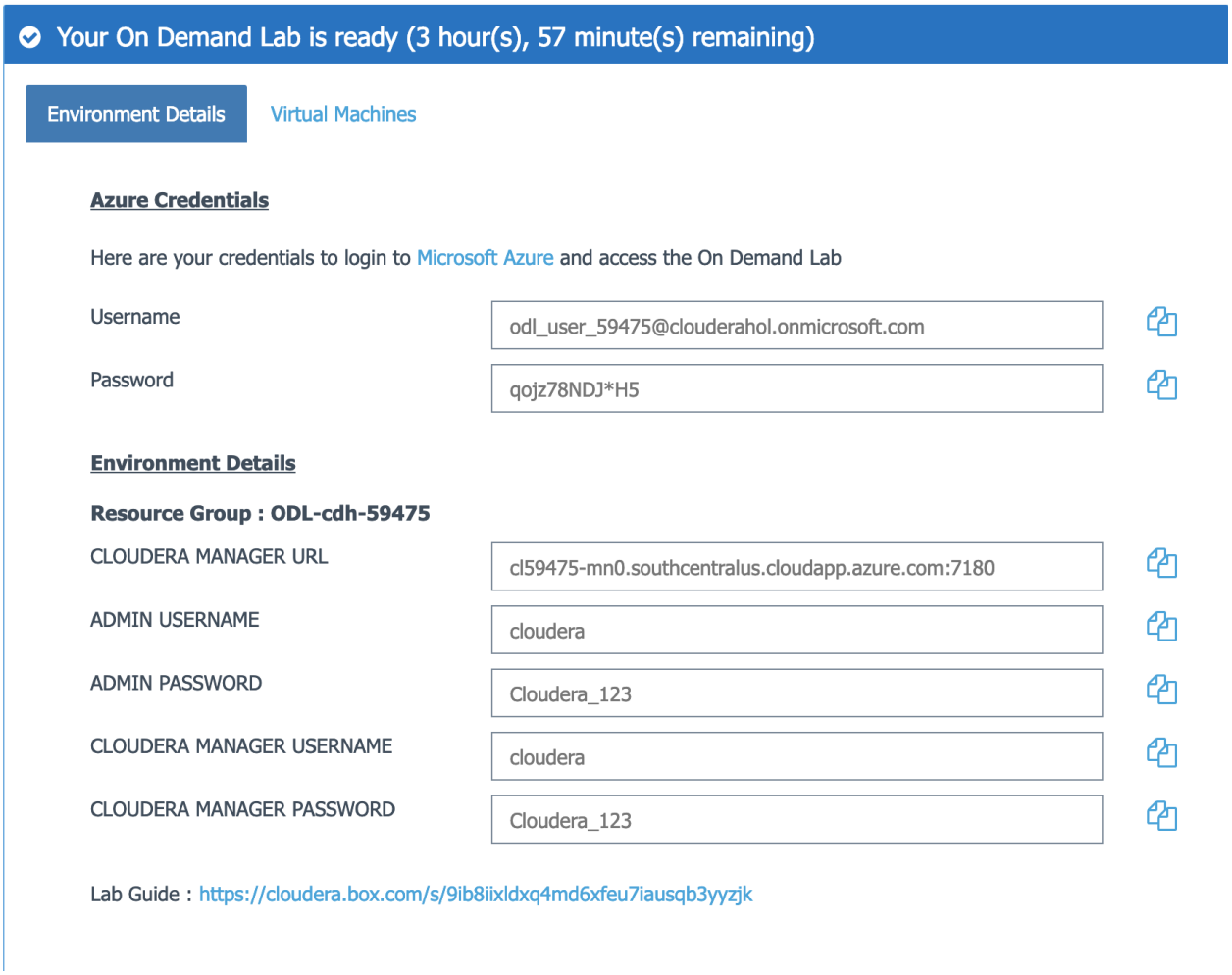
Check the email:



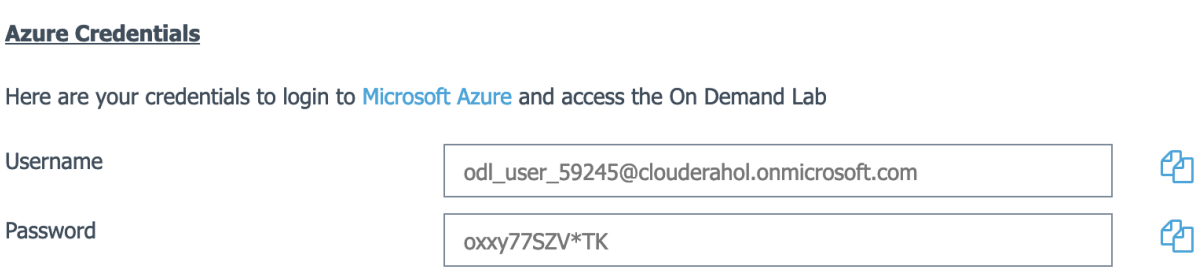
Then click on **Launch Lab** button



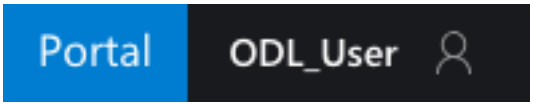
Click again on the **LAUNCH LAB** button and you should get your Azure credentials and cluster details:

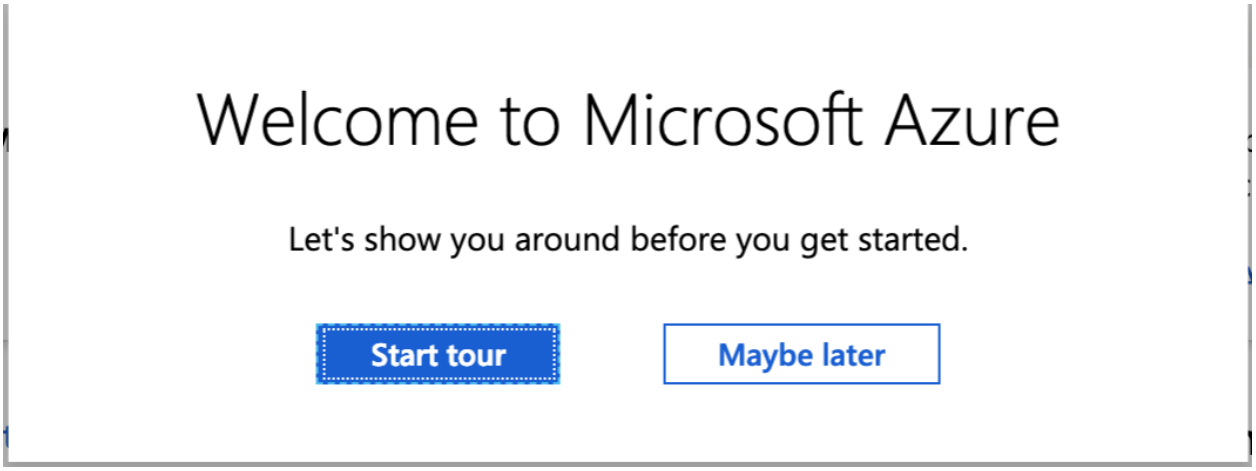


Use your temporary Azure credentials to log into Azure at the following URL = https://azure.microsoft.com/en-us/



Click on **Sing-in** and provide the Azure credentials and once logged-in you will see the Azure in the top right hand corner Portal and ODL_User
Click on **Portal**

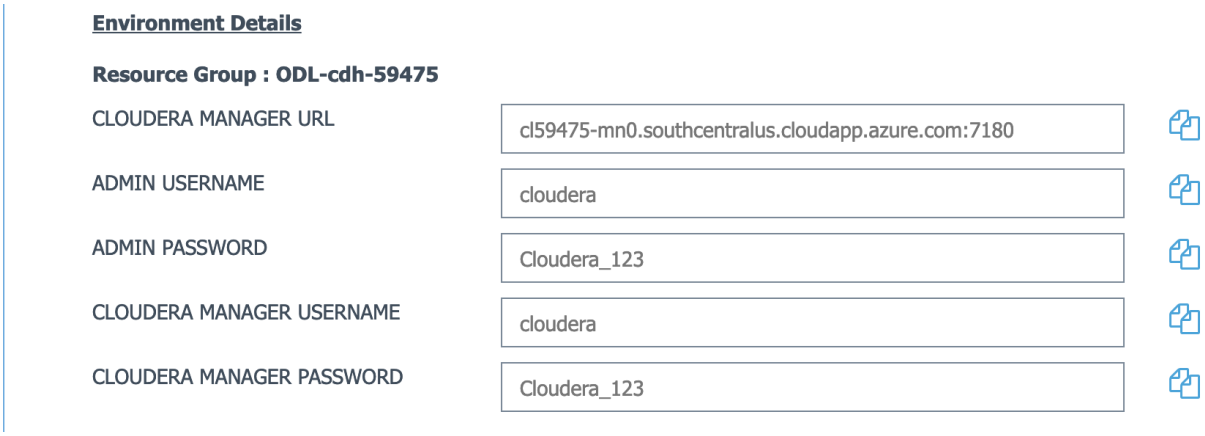




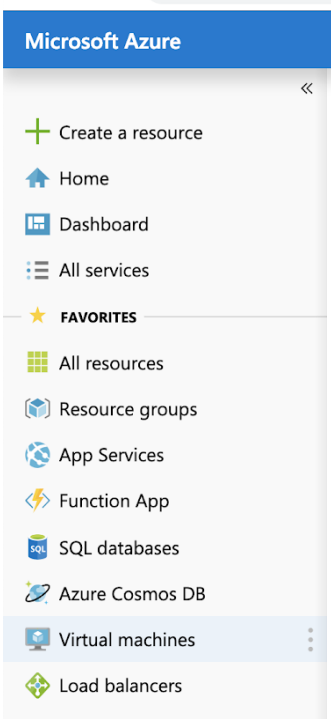
Click **Maybe later** button.

Lab 2 - Access the pre-provisioned Cloudera cluster

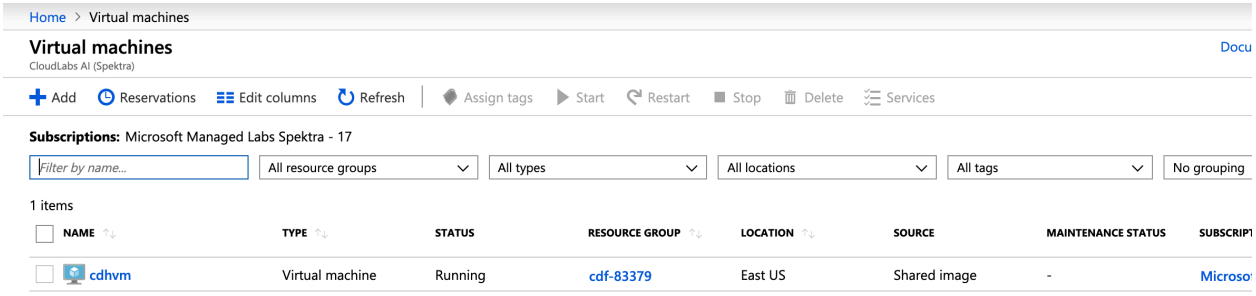
The details of the preprovisioned cluster are available here:



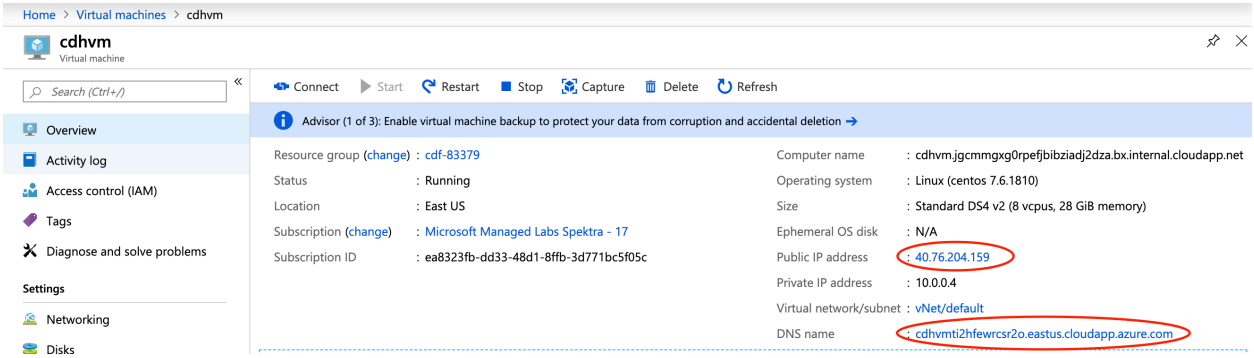
In the Azure portal check the VMs created by clicking on the V**irtual machines** link



This is your pre-provisioned cluster with a single-node cluster:


Home > Virtual machines
Virtual machines
CloudLabs AI (Spektra)
Docu
Add
Reservations
Edit columns
Refresh
Assign tags
Start
Restart
Stop
Delete
Services
Subscriptions: Microsoft Managed Labs Spektra - 17
Filter by name...
All resource groups
All types
All locations
All tags
No grouping
1 items
NAME
TYPE
STATUS
RESOURCE GROUP
LOCATION
SOURCE
MAINTENANCE STATUS
SUBSCRIPT
cdhvm
Virtual machine
Running
cdf-83379
East US
Shared image
-
Microso

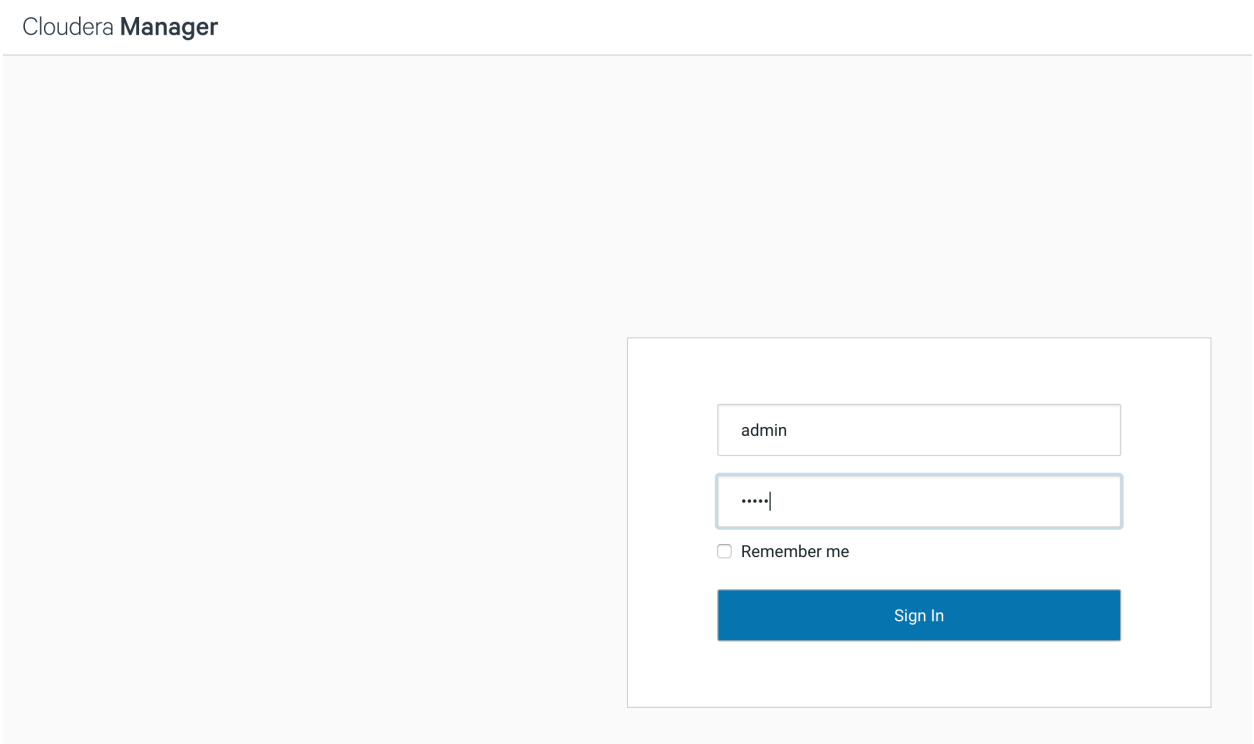
Click on the Virtual machine name (cdhvm) to see its public ip address as well as its private ip address and its dns name:



Write down the ip address (or the dns name) for later reference in the workshop.

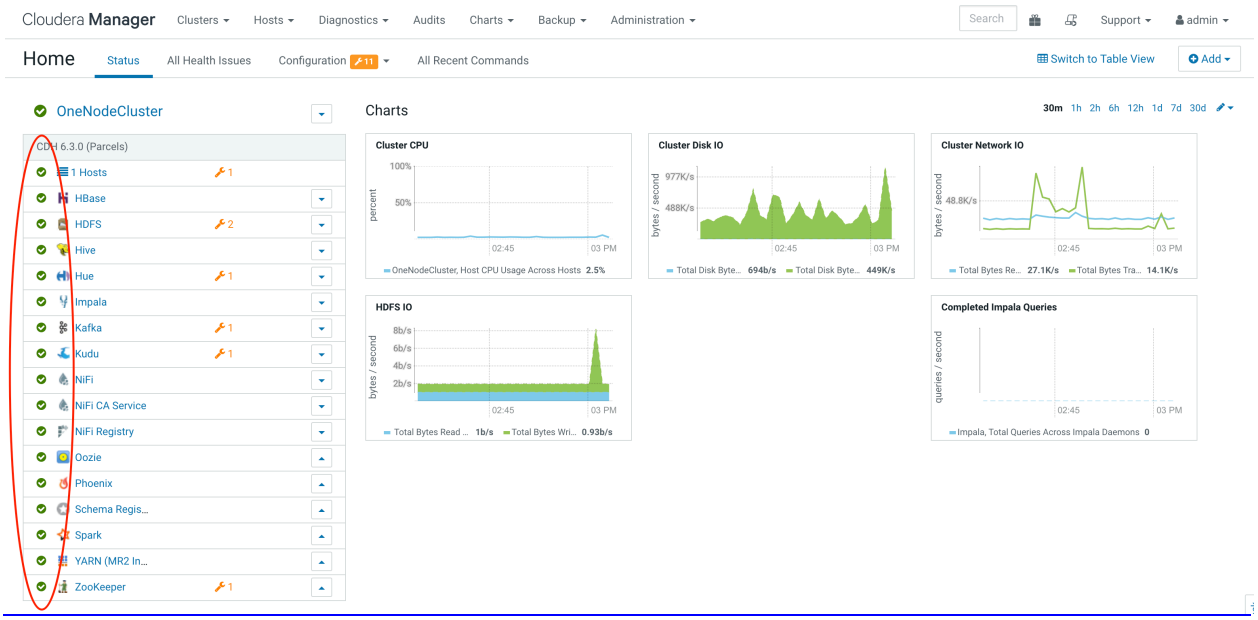
Check of all services of your Cloudera Data Hub cluster are up-and-running.
Navigate to the Cloudera Manager UI using the url (change the ip address in the provided example with your ip address):
http://40.76.204.159:7180

The link will take you to the Cloudera Manager login page:



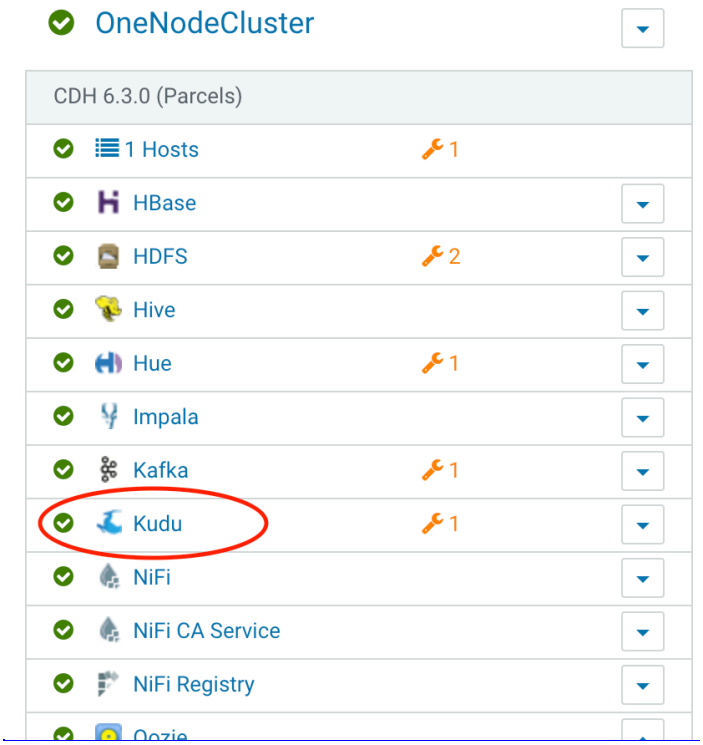
Login to Cloudera Manager using user ‘admin’ with the password ‘admin’.

After login you will see all services running on the cluster. If all services are 'green', you are ready to proceed. If some are red, you may need to wait a bit longer before you can proceed.

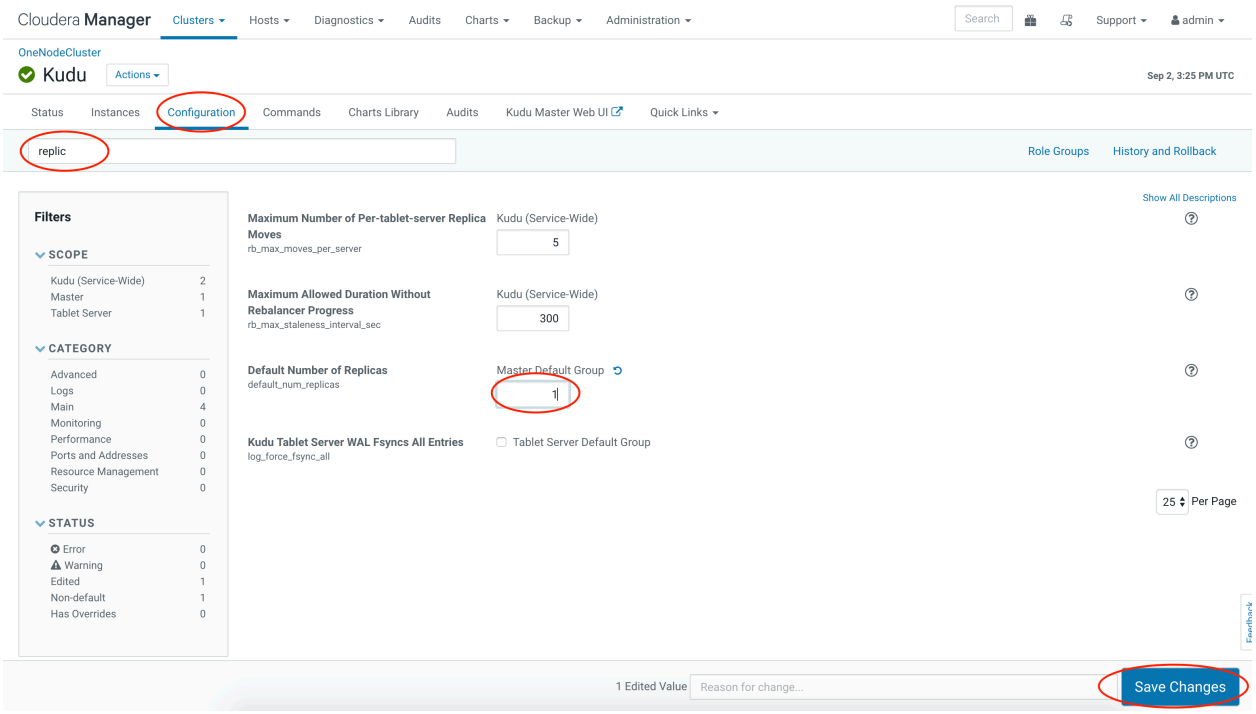


Change the replication factor in Kudu:
In this lab, we will write data to Kudu. By default, data in kudu will be replicated 3 times on different tablet servers, to support High Availability. But since this test-cluster only has 1 node, 3 times replication cannot be used. By switching back to 1 replica, we will be able to use kudu in this workshop. Change the replication factor as follows:

Go your Cloudera Manager interface, and select the Kudu service:

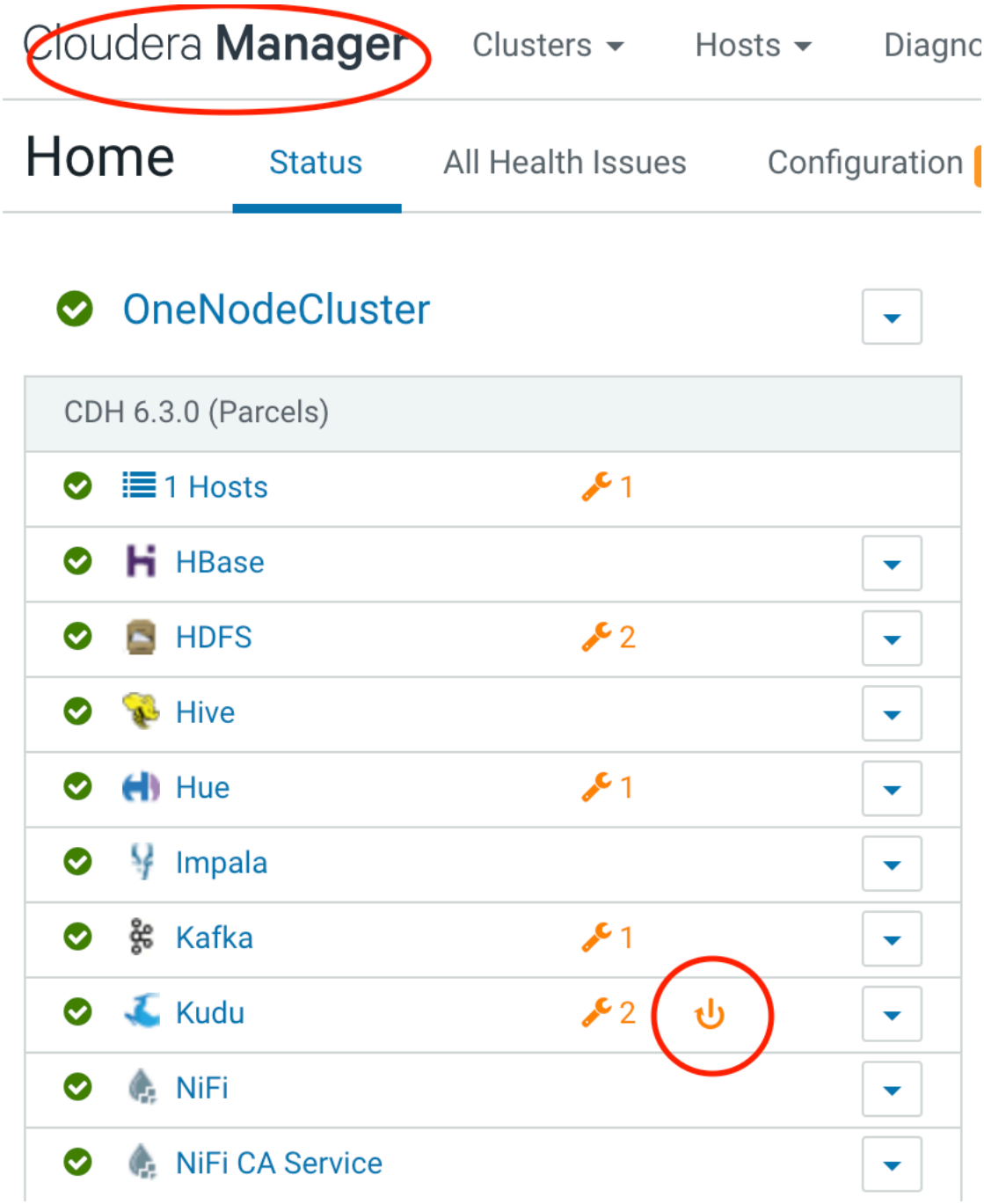


Go to the 'Configuration' page, search for 'replic', and change the number of replica's to 1. Hit 'Save Changes' to commit your change.

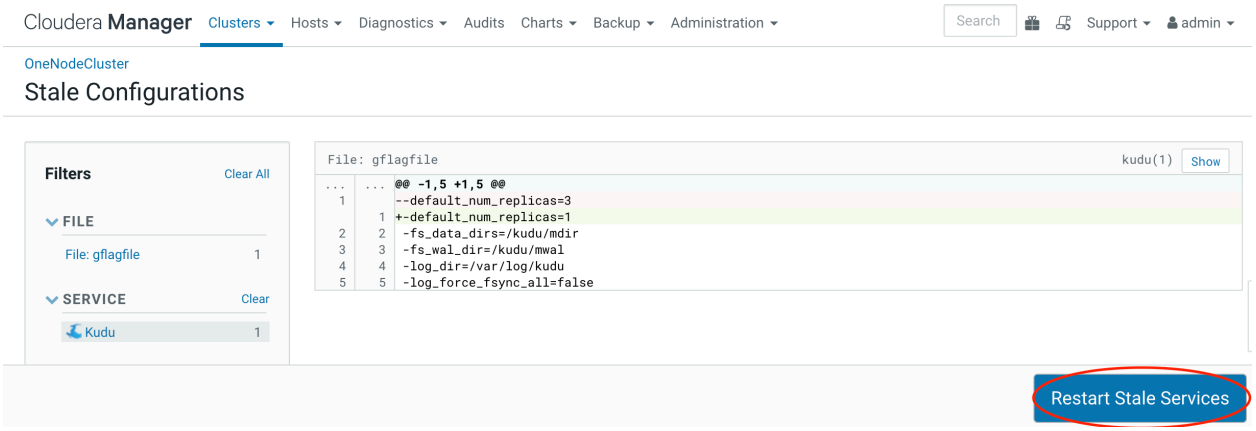


After the change, the service needs to be restarted.

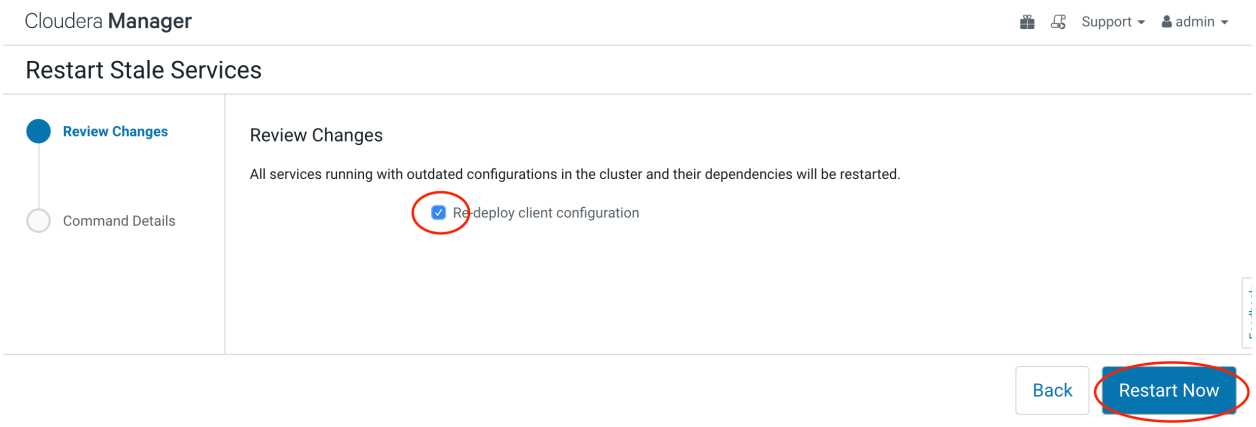
Navigate to the Cloudera Manager homepage, and click on the restart service buttion behind the Kudu service:



Hit the 'Restart Stale Services' button.

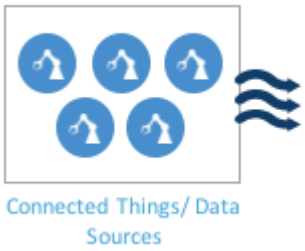


Select the 'Re-deploy client configuration' checkbox, and hit 'Restart Now' button.



Wait till the restart of the Kudu service is completed and all services on the cluster are running again.

Lab 3: Get the Connected Things/Data Sources



For the purpose of this demo, for sake of simplicity, we will simulate the remote, connected things. Here you will see how to create a NiFi pipeline in order to simulate generic remote Things that generate time series data.

The remote, connected Things that we are going to simulate are sensors defined by the following 3 fields:

- Sensor ID
- Sensor Value
- Timestamp

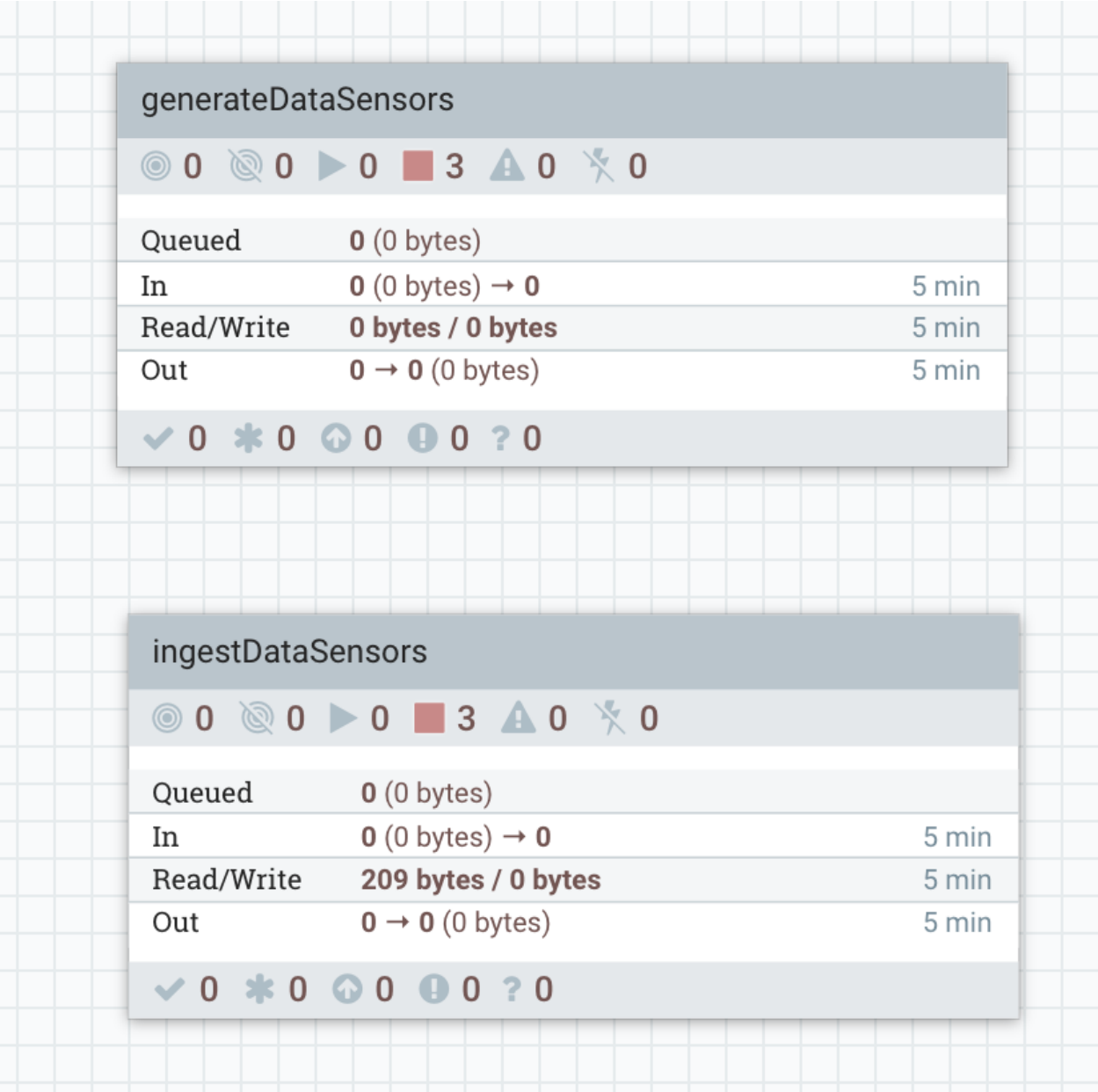
Where the Sensor ID is the unique identifier of the sensor, and for our demo we can assume that there are 10.000 sensors; value and timestamp are the timeseries reading.
You are free to model a more complex remote Things, but for the rest of this exercise we will consider the above 3 fields for the remote Thing.

Once that we have conceptually modelled the remote Thing, now we are going to build a pipeline in NiFi that will act as a simulator for those 10.000 remote Things; it generates the random sensor reading and publishes the readings to a Kafka topic.

With NiFi we are going to create two flows with these these goals:

- One flow that will generate the time-series sensor reading, we are going to call *generateDataSensors*
- A second flow that will read the time-series sensor reading and store into Kudu, we are going to call *ingestDataSensors*

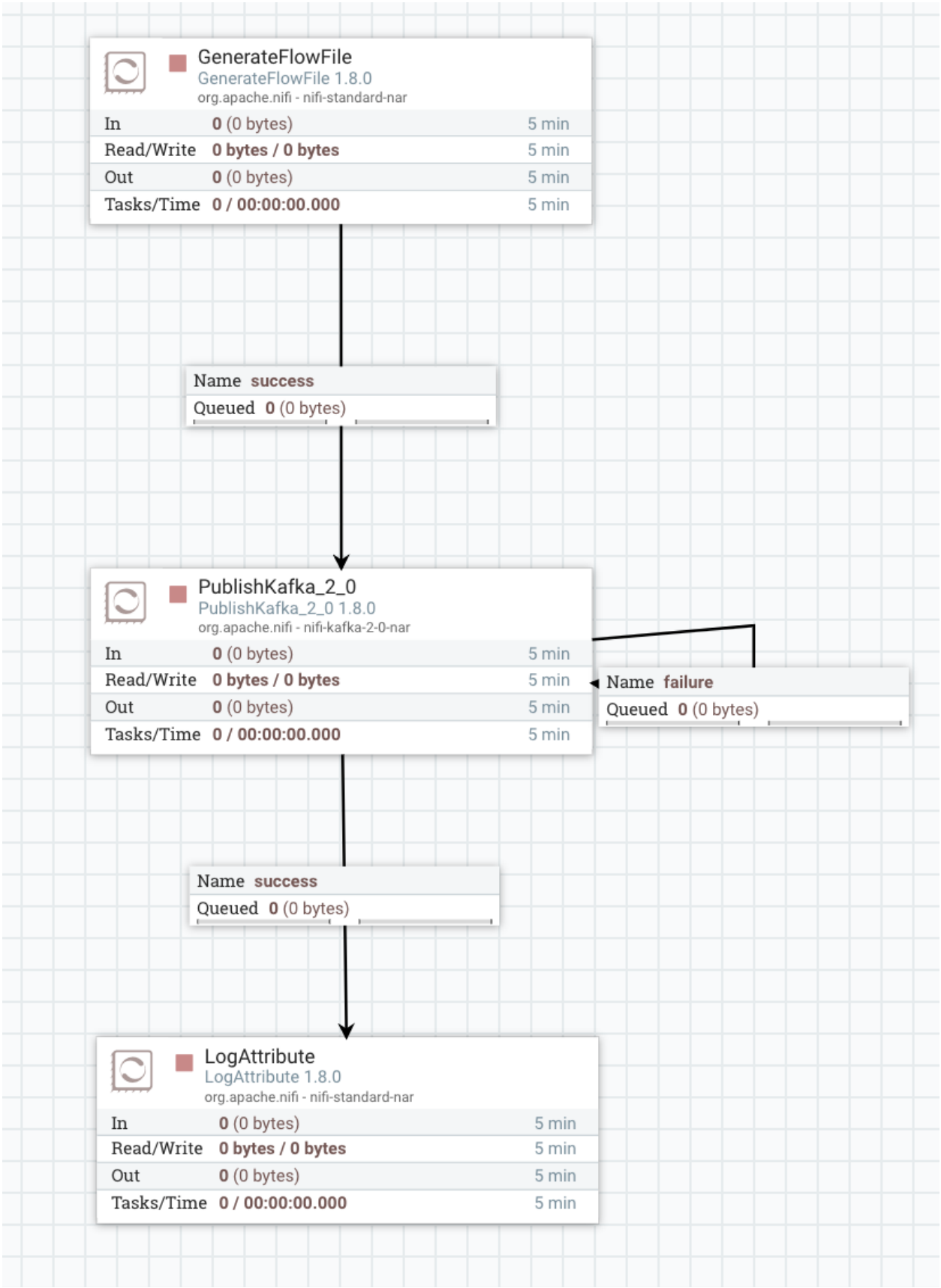
At the high level we will have the following:



Let's start with the first flow.

generateDataSensors flow

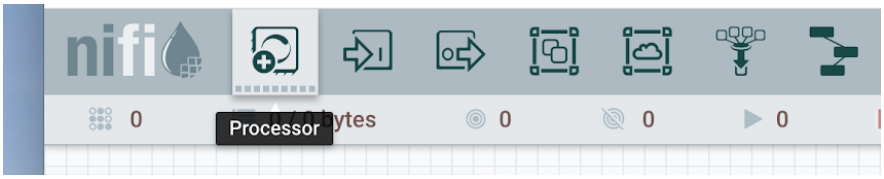
This flow is going to look like the following picture

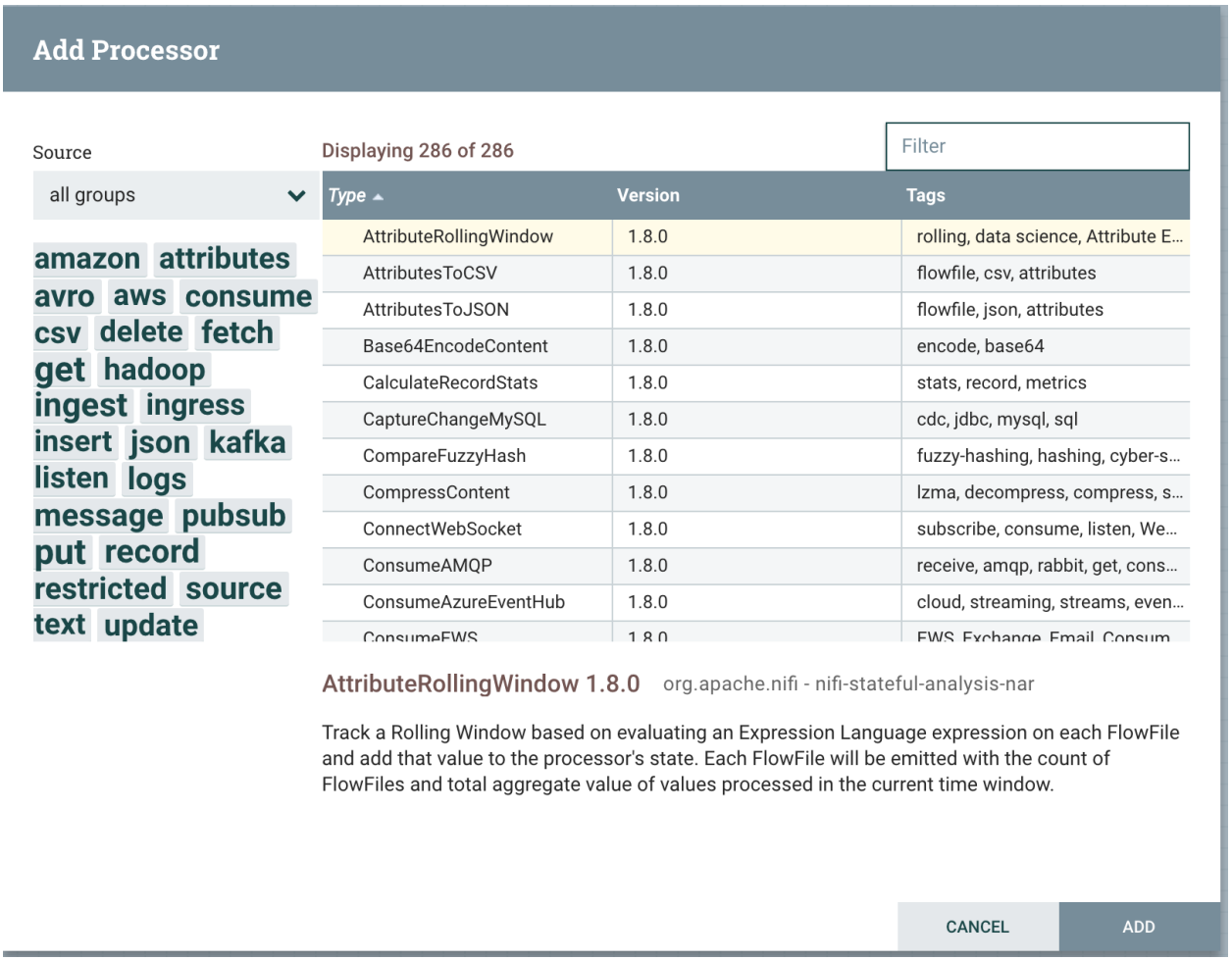

GenerateFlowFile
GenerateFlowFile 1.8.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
PublishKafka_2_0
PublishKafka_2_0 1.8.0
org.apache.nifi - nifi-kafka-2-0-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name failure
Queued 0 (0 bytes)
Name success
Queued 0 (0 bytes)
LogAttribute
LogAttribute 1.8.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Follow these steps to create the above flow in your NiFi instance.

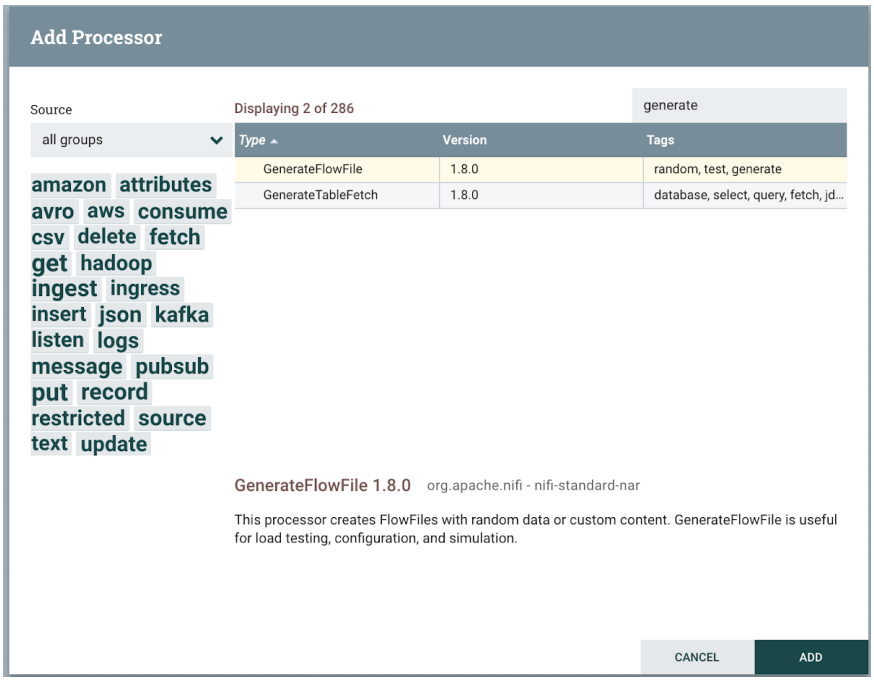
Login to the NiFi web UI (use your ip address instead of this example)
http://40.76.204.159:8080/nifi/

On the top-left corner, click on the **Processor** icon and drag it into the canvas:

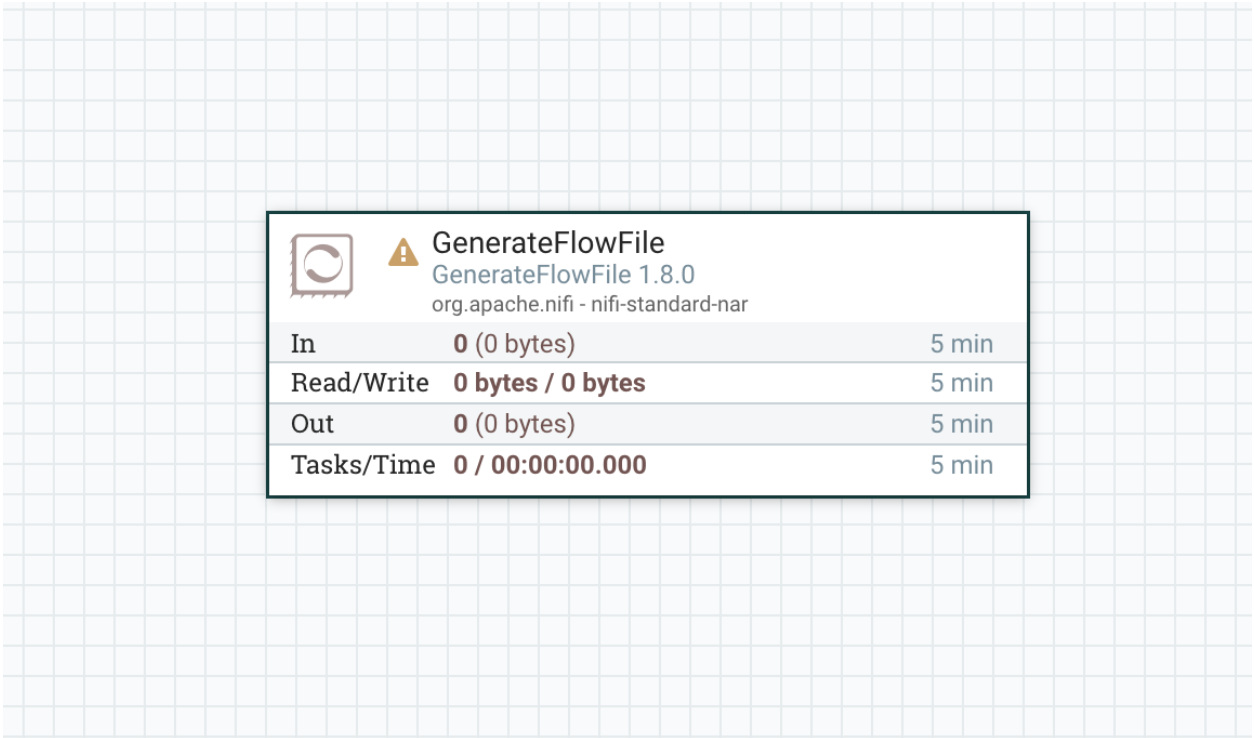




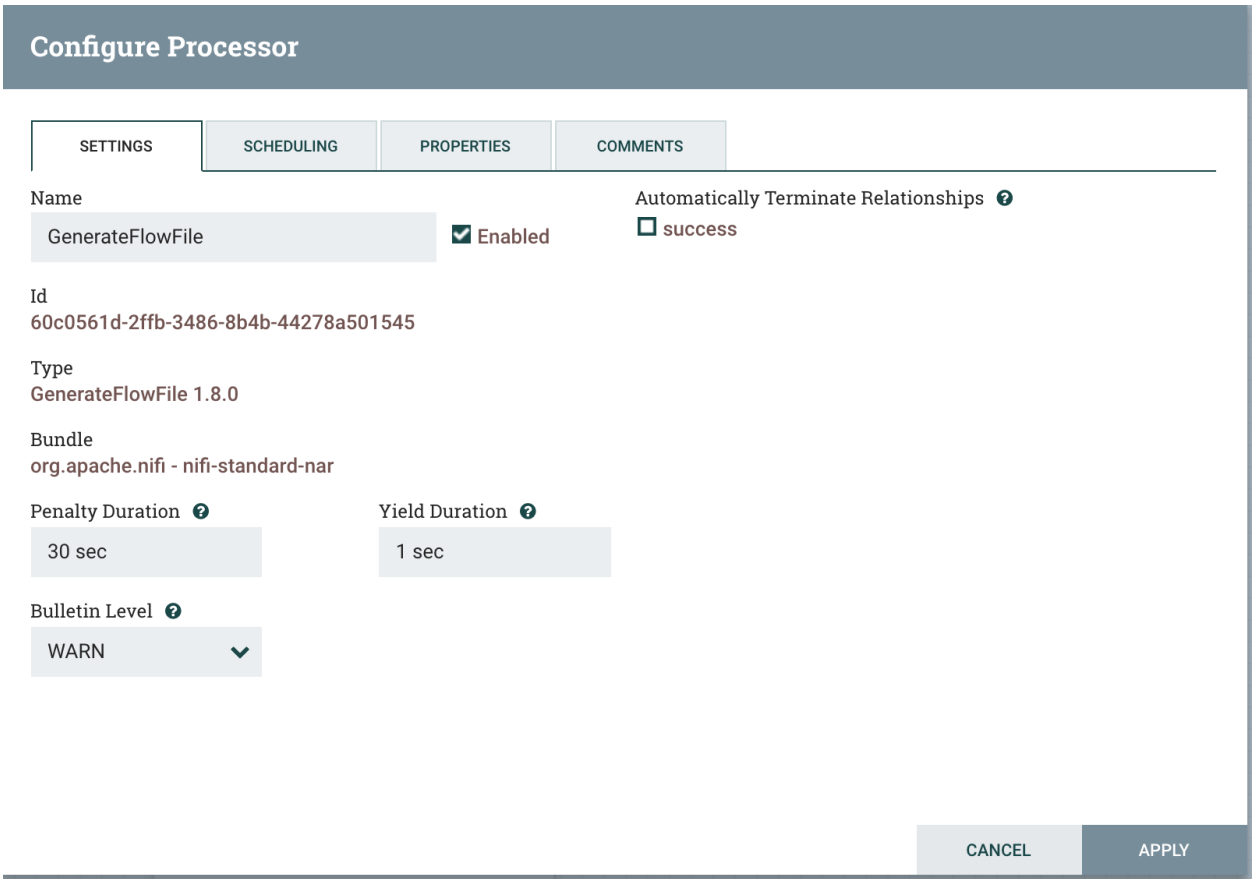
In the **Filter** field type *generate* and select **GenerateFlowFile** and click **ADD**



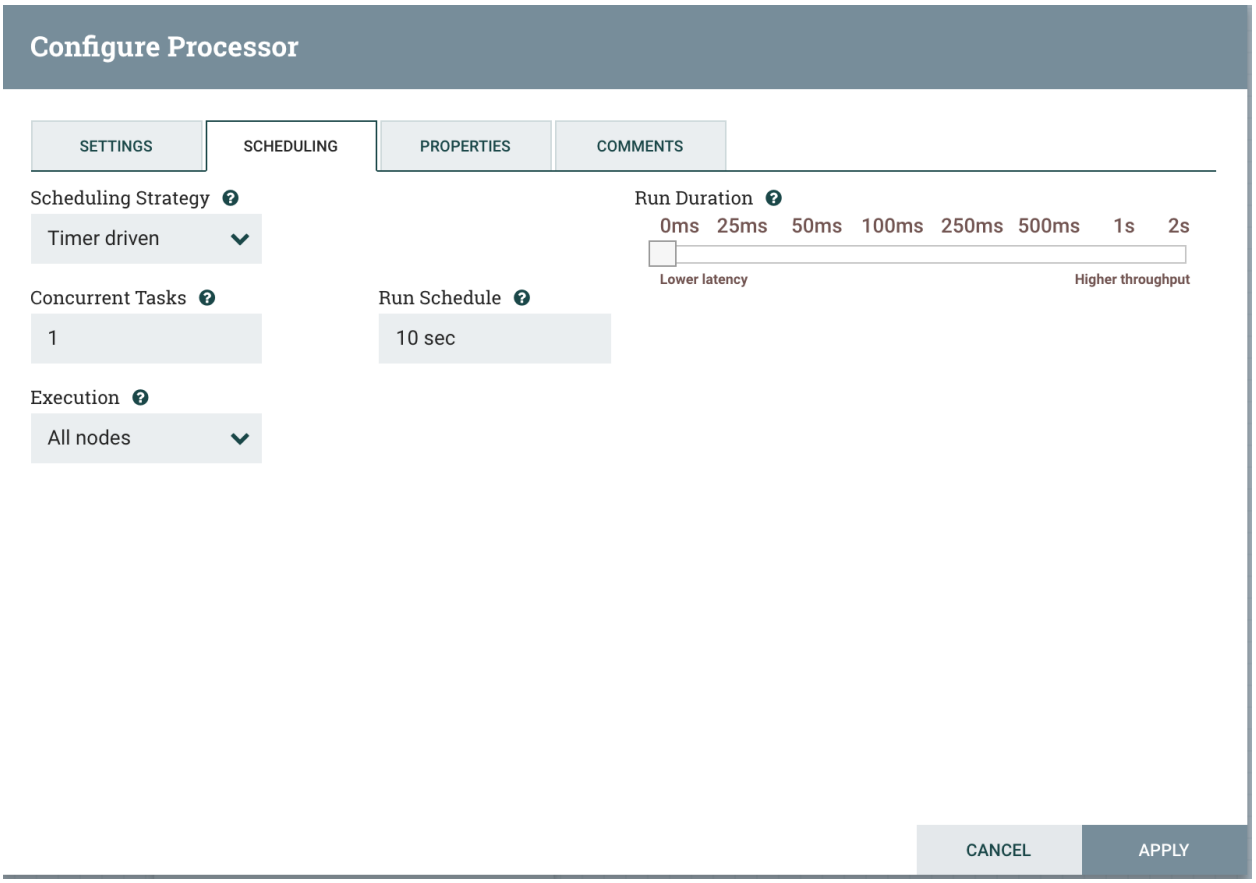
As a result you will get in the canvas the following component:



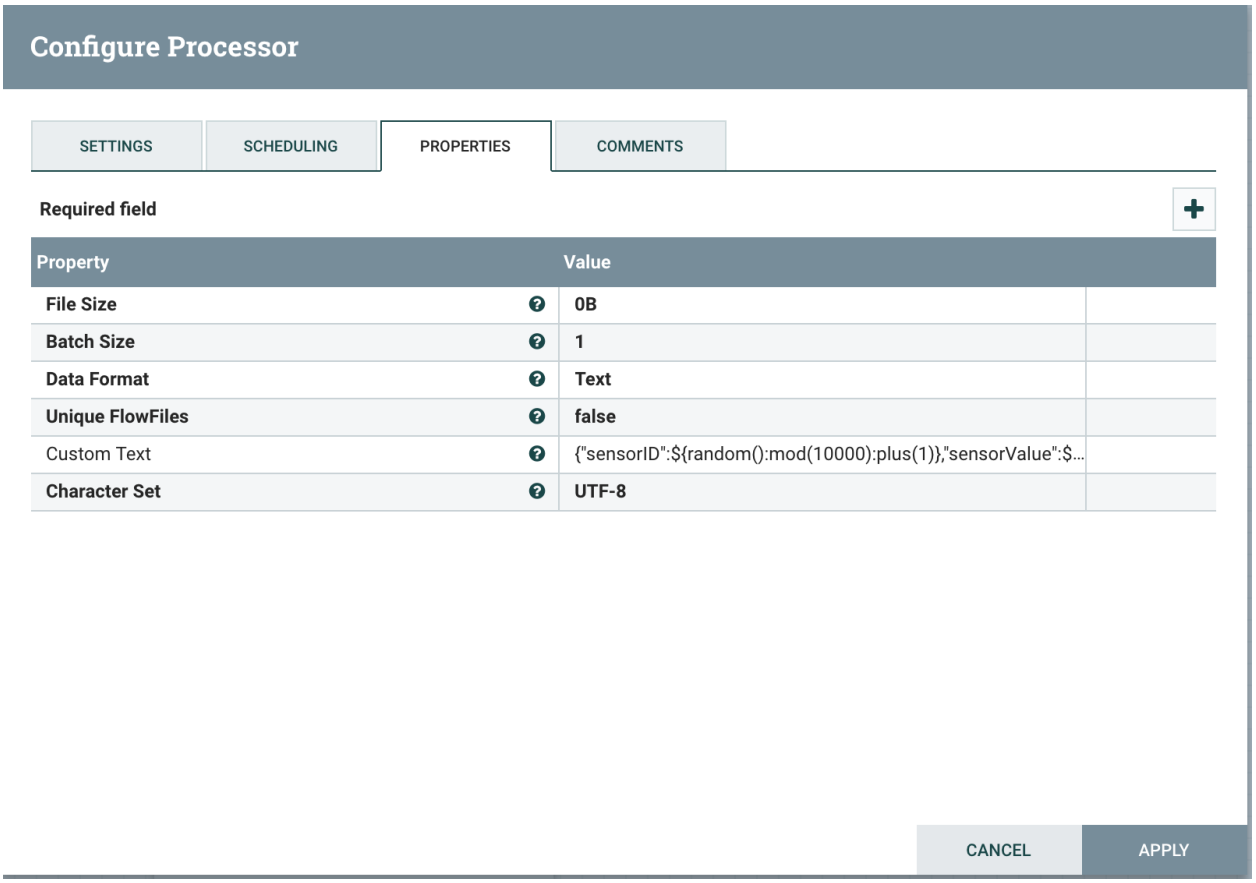
We are going to use this component in order to generate random time-series sensor reading. Double click on this component:



Go to the **SCHEDULING** tab:



And change the **Run Schedule** to 10 sec, in order to generate a sensor reading every 10 seconds. Then go to the **PROPERTIES** tab:



And in the **Value** of the **Custom Text** field add the following:

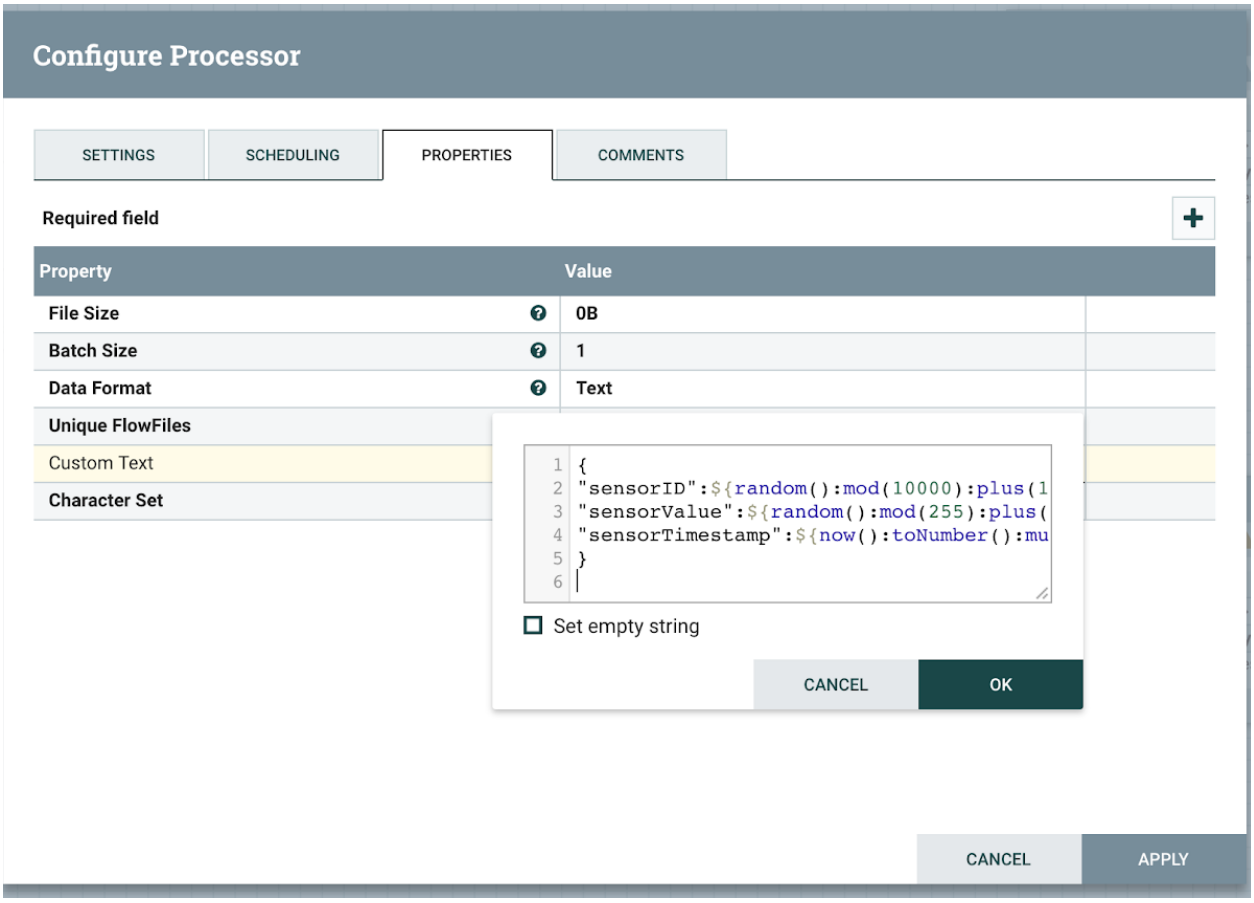
```
{
        "sensorID":${random():mod(10000):plus(1)},
        "sensorValue":${random():mod(255):plus(1)},
        "sensorTimestamp":${now():toNumber():multiply(1000)}
}
```

This expression will generate JSON messages like:

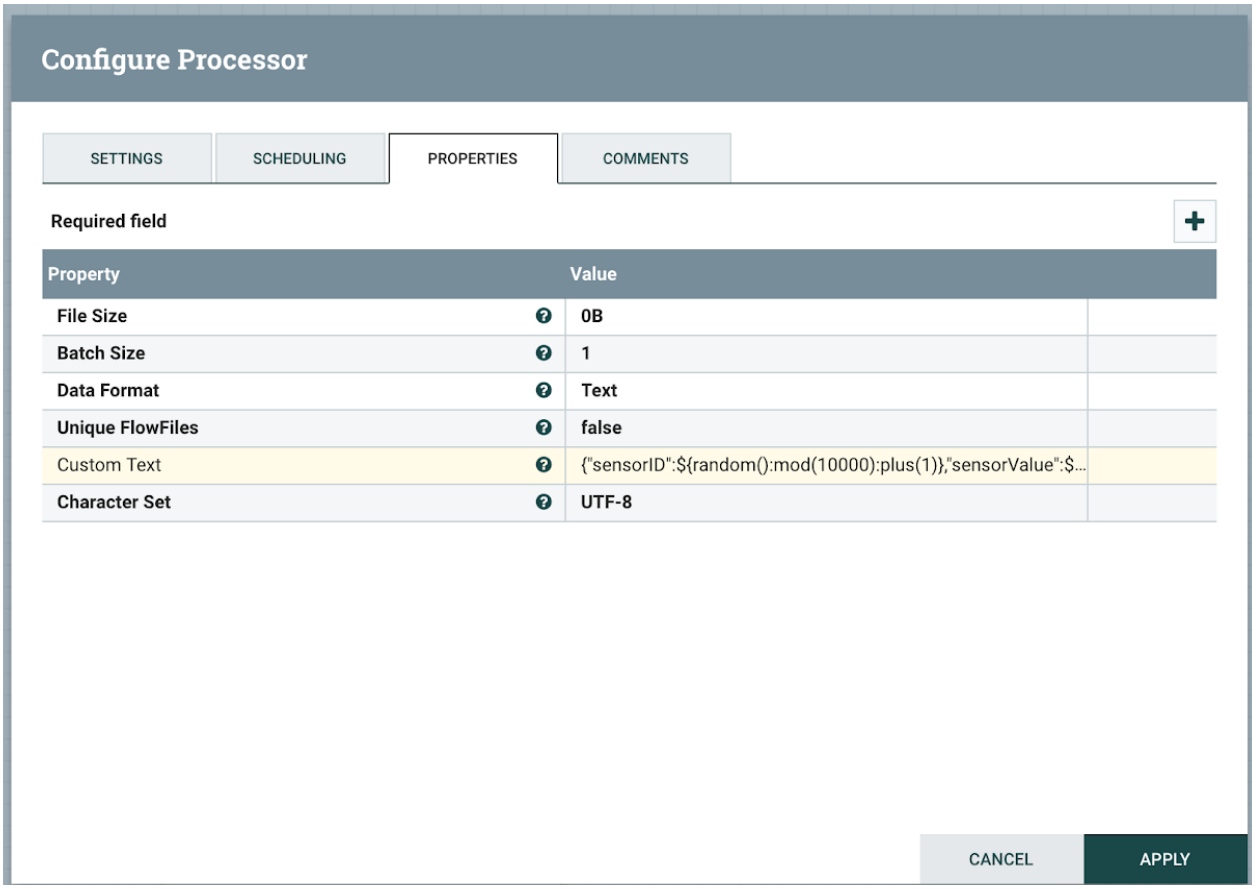
```
{
        "sensorID":5172.0,
        "sensorValue":16.0,
        "sensorTimestamp":1.547053587308E12
}
```

Basically we are using the NiFi Expression Language to generate sensor reading with the following characteristics:

- A sensor identifier with a value from 1 to 10000
- A sensor reading with a value from 1 to 255
- And a timestamp

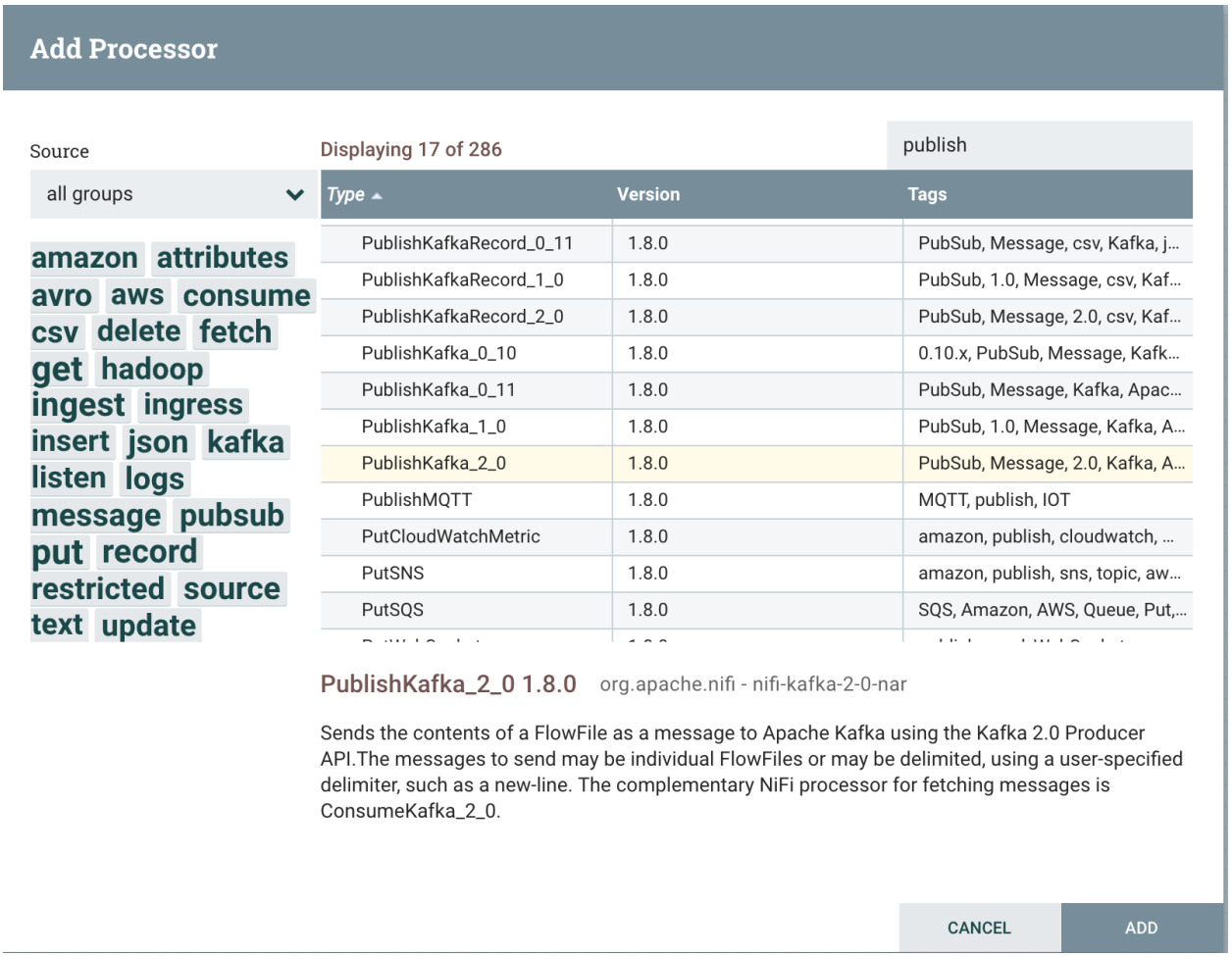


And click **OK** and then click **APPLY**:

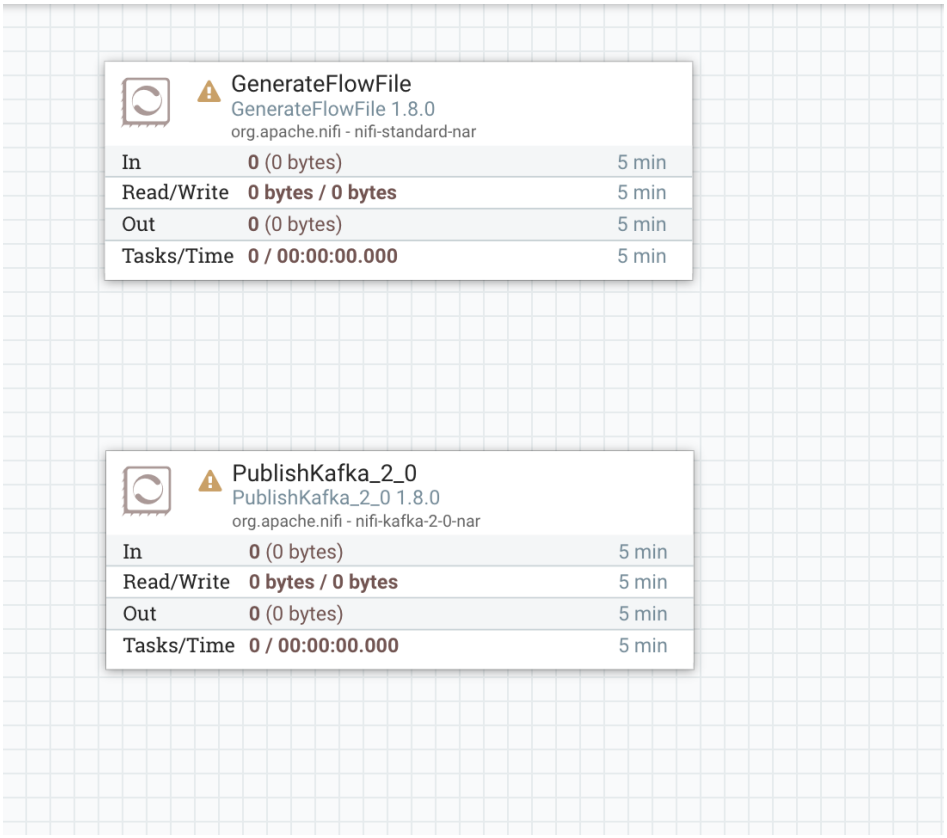


Now that we have a component that generates random reading, when can proceed with our flow and add the component that will be responsible to publish the message to a Kafka topic.

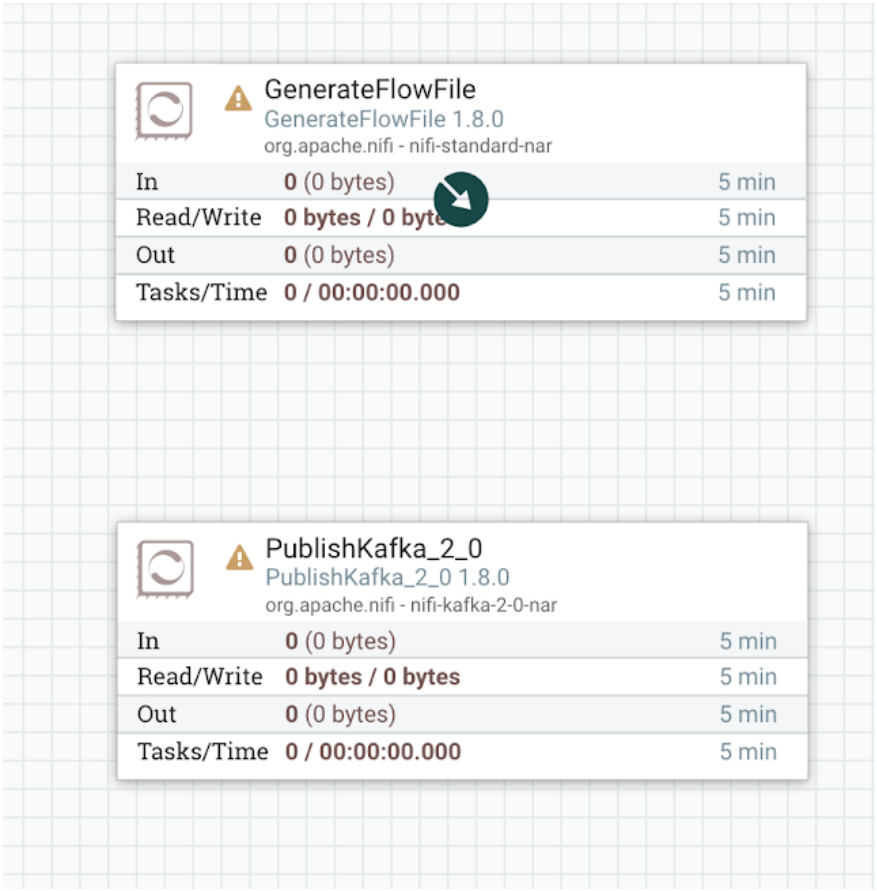
Drag & Drop again the **Processor** icon into the canvas, and select PublishKafka_2_0



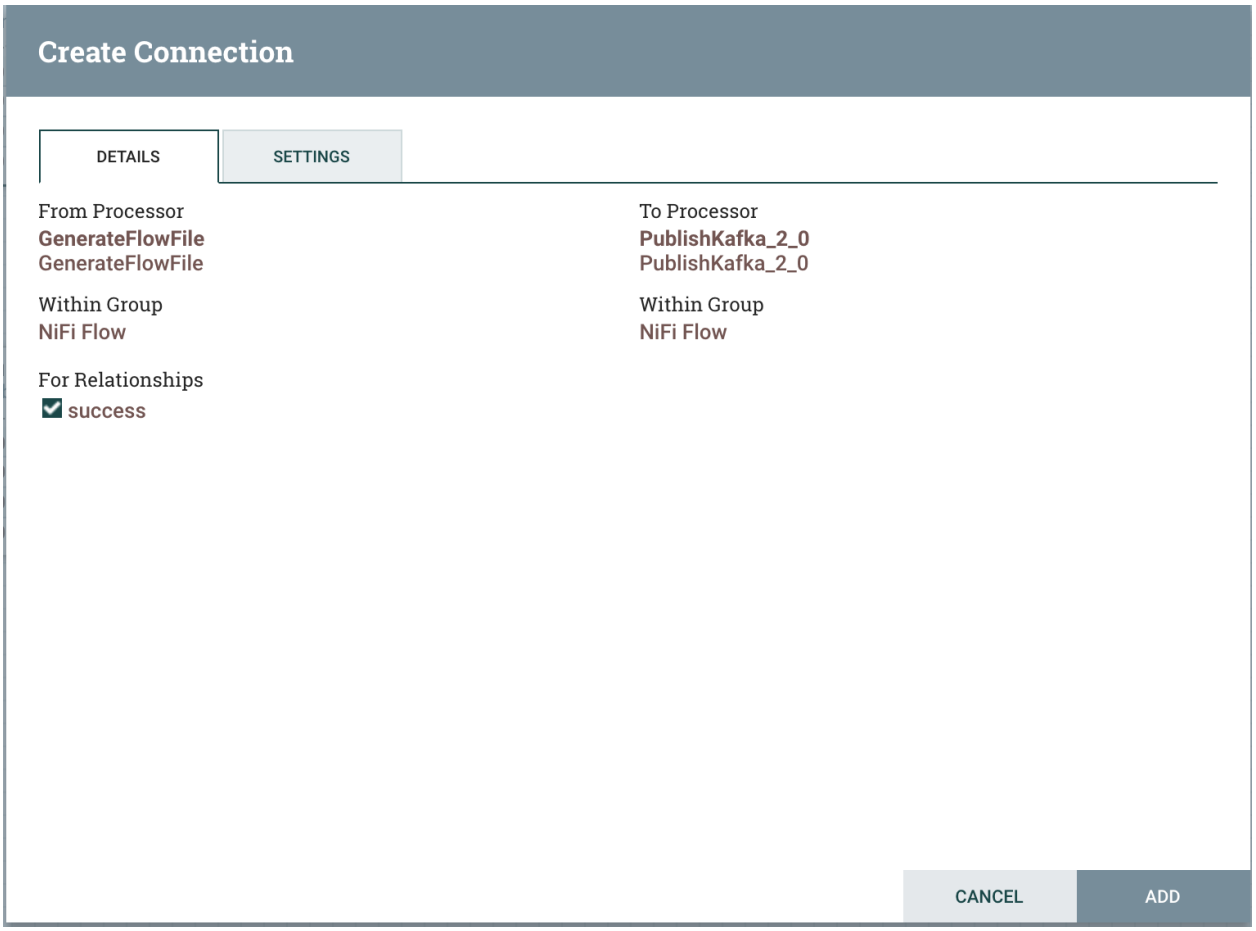
Then click **ADD**.
Now your canvas should look like this:



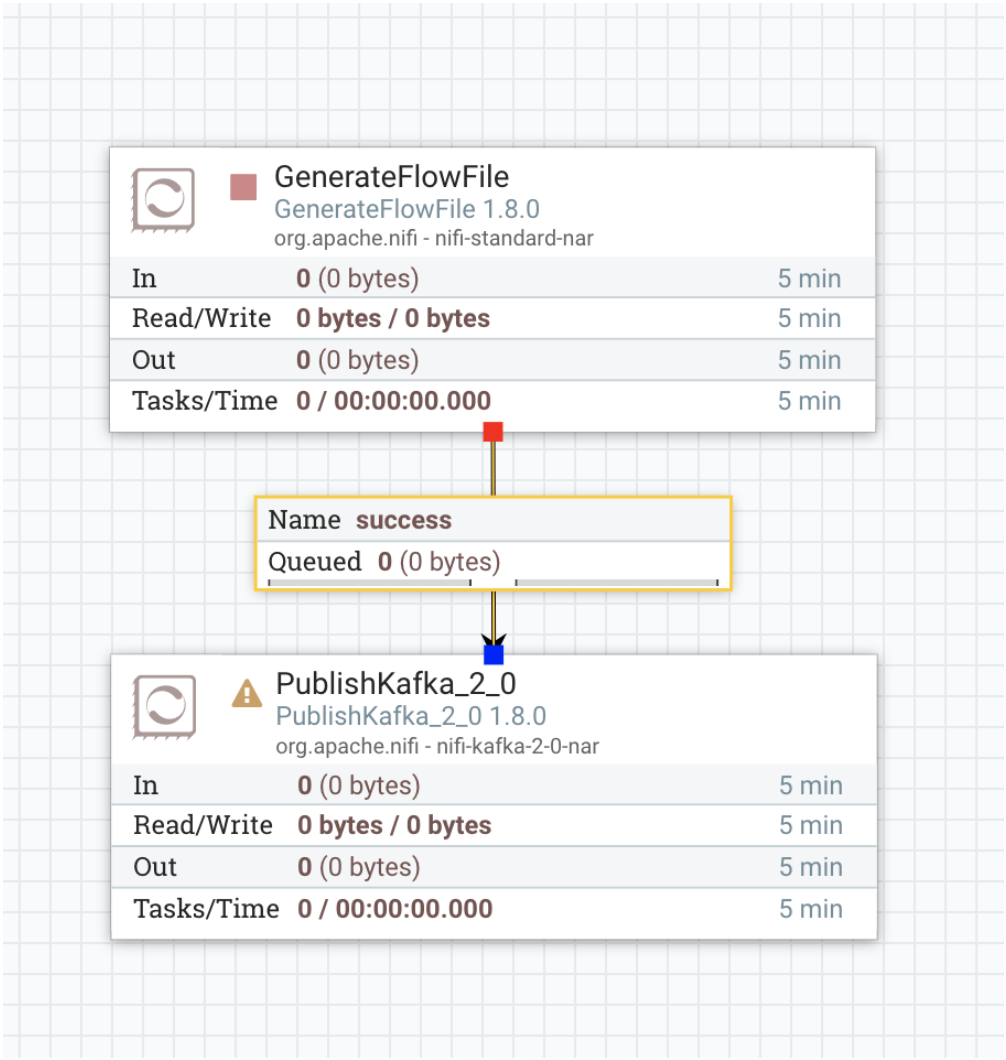
Now, with the mouse go over the **GenerateFlowFile** step and you should see an arrow icon:



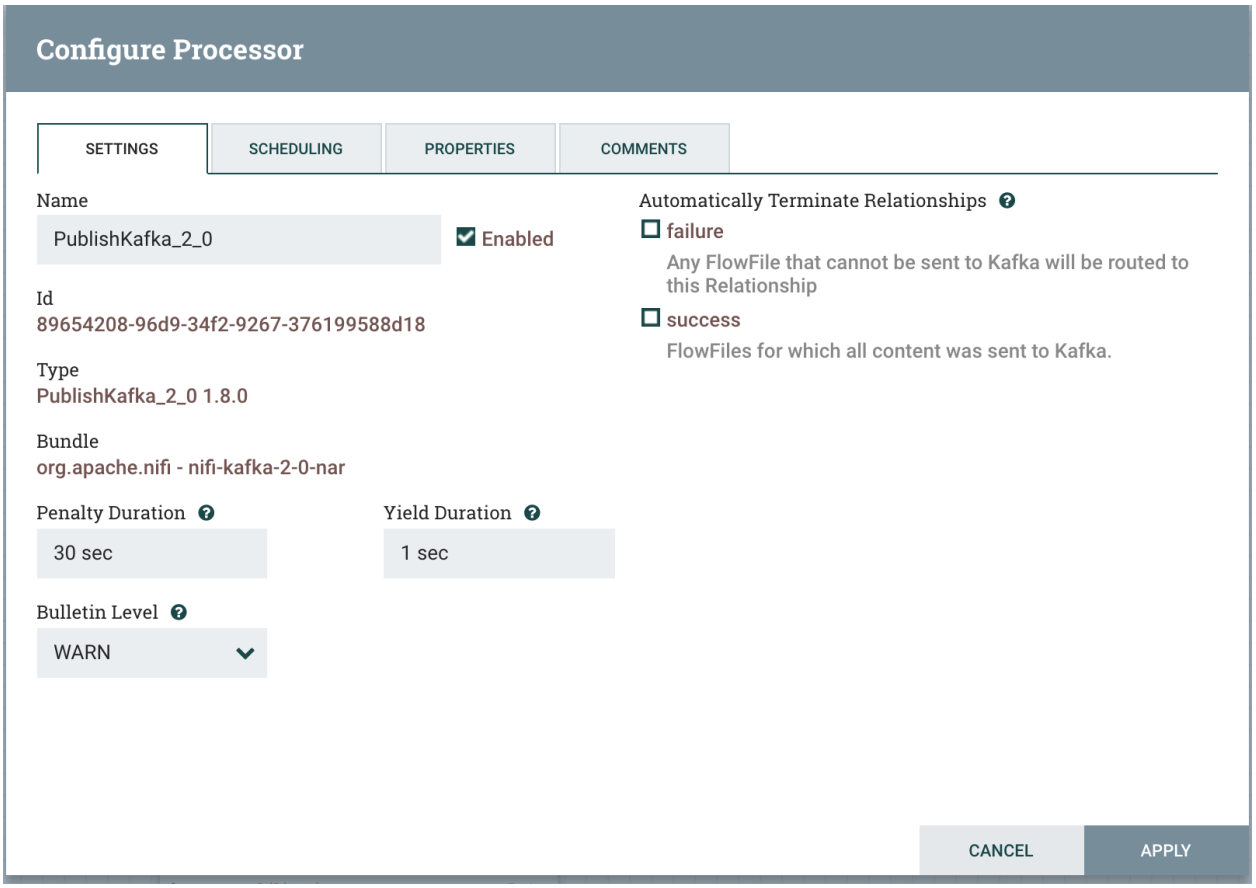
Drag & Drop that icon to the **PublishKafka_2_0**, in order to connect the two steps:



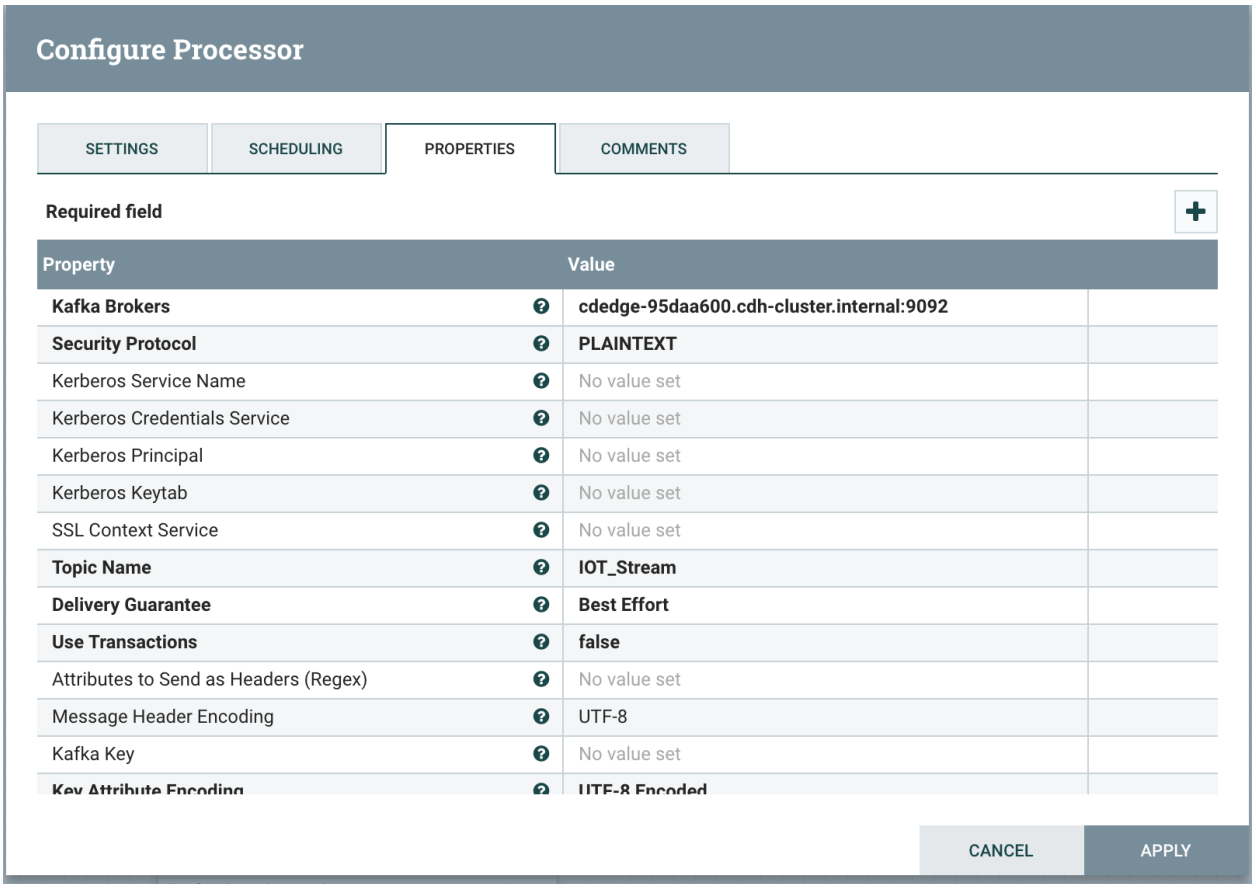
Then click **ADD**, and you should see something like this:



Now click on the **PublishKafka_2_0** in order to configure it.



Go to the **PROPERTIES** tab:

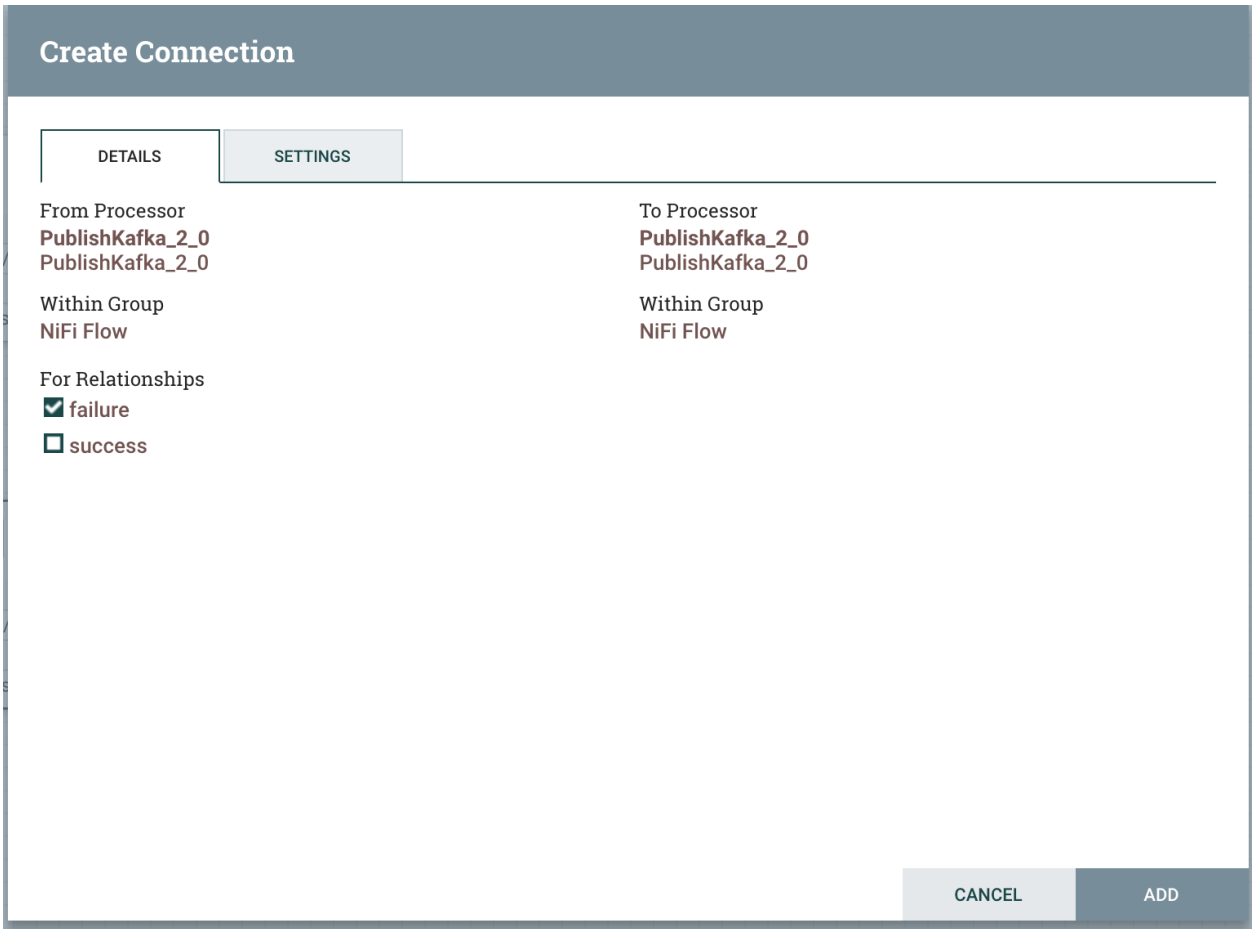


And add the value for the following properties:

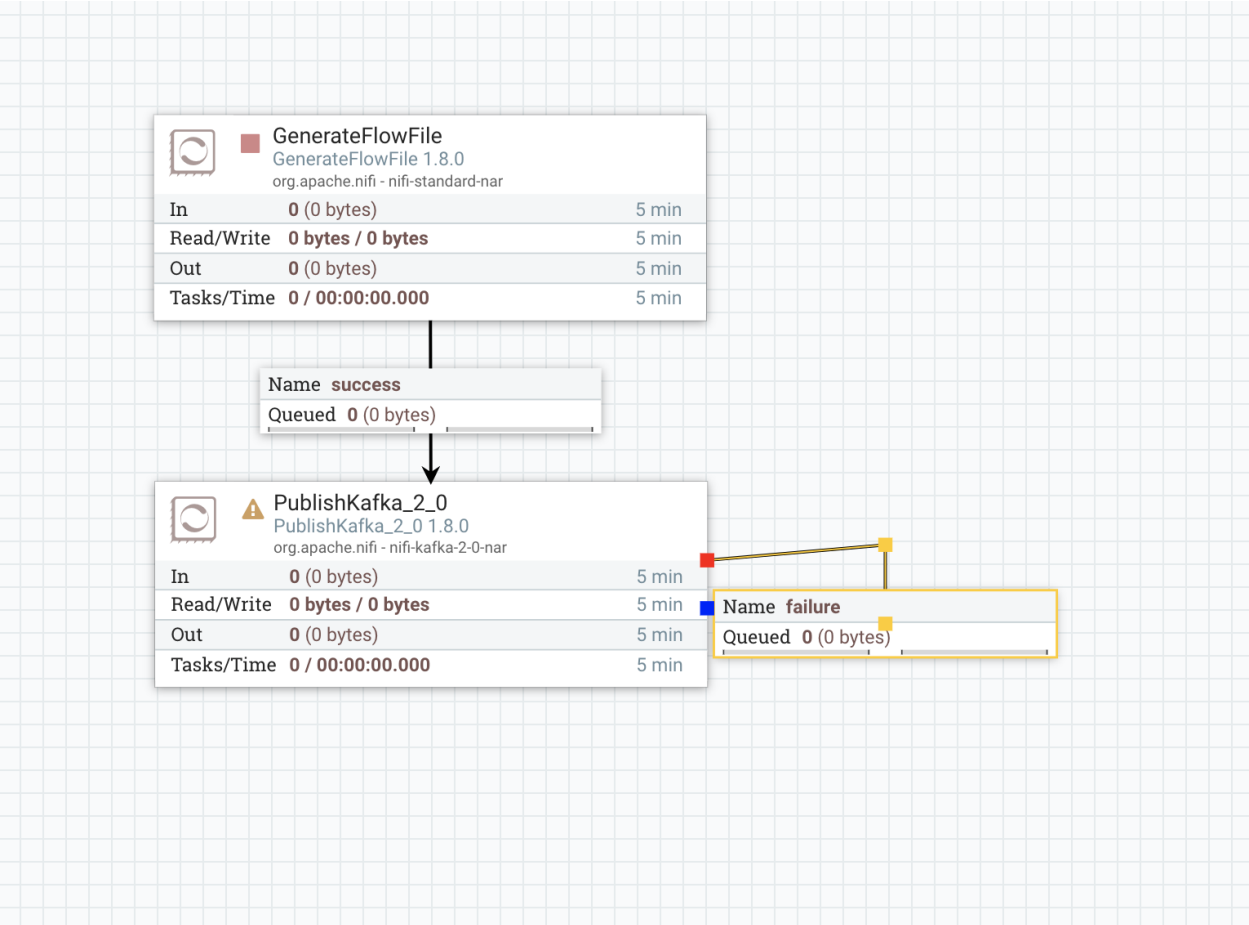
- **Kafka Brokers**: <your-private-ip>:9092 (in my case 10.0.0.4:9092)
- **Topic Name**: IOT_Stream
- **Use Transactions**: false

Once done, click **APPLY**.

In case of something goes wrong when publishing to kafka topic, we want to catch it in order, so go over the **PublishKafka_2_0** click on the arrow icon and drag it outside the component and terminate it into the component itself. The Connection box pop-up, check the **failure** box as described below:

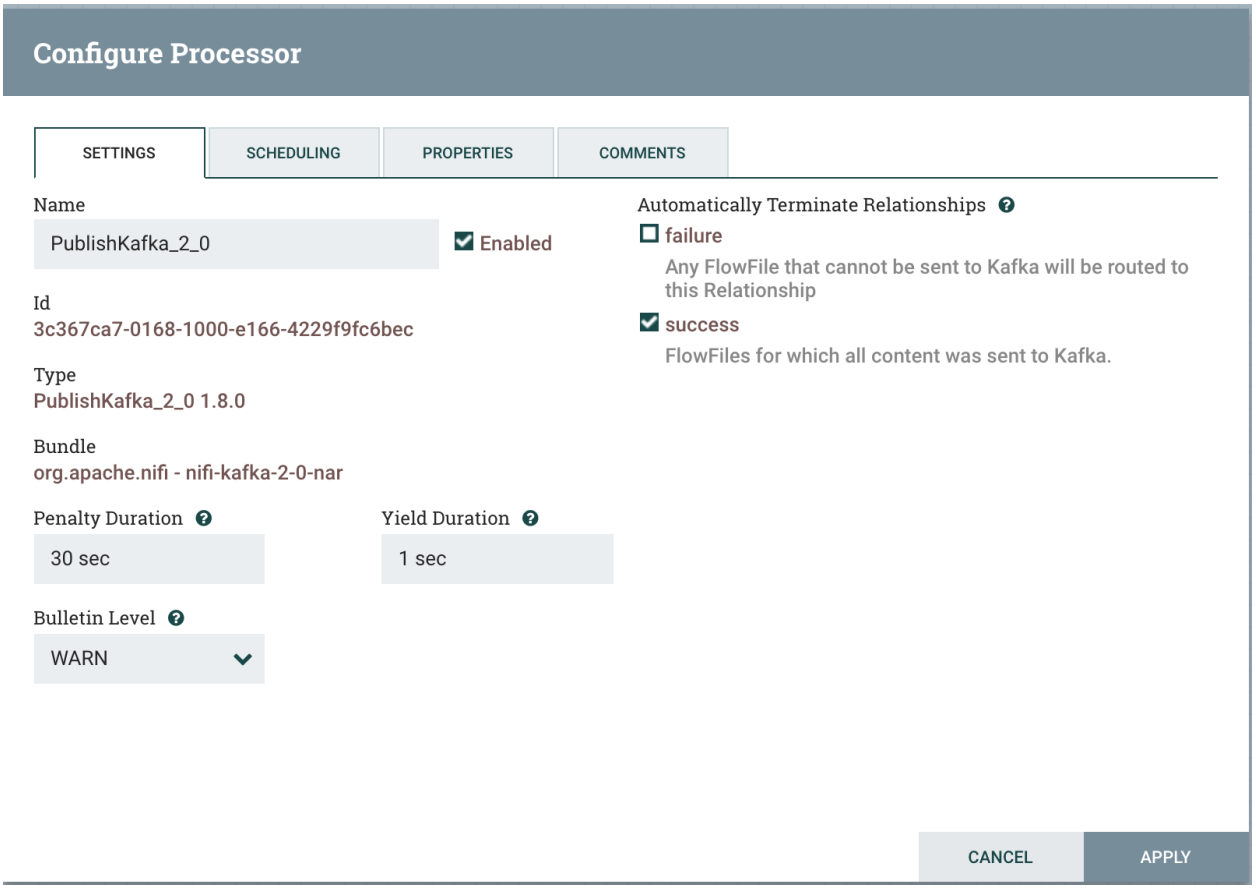


Then click **ADD**, and you should get this:

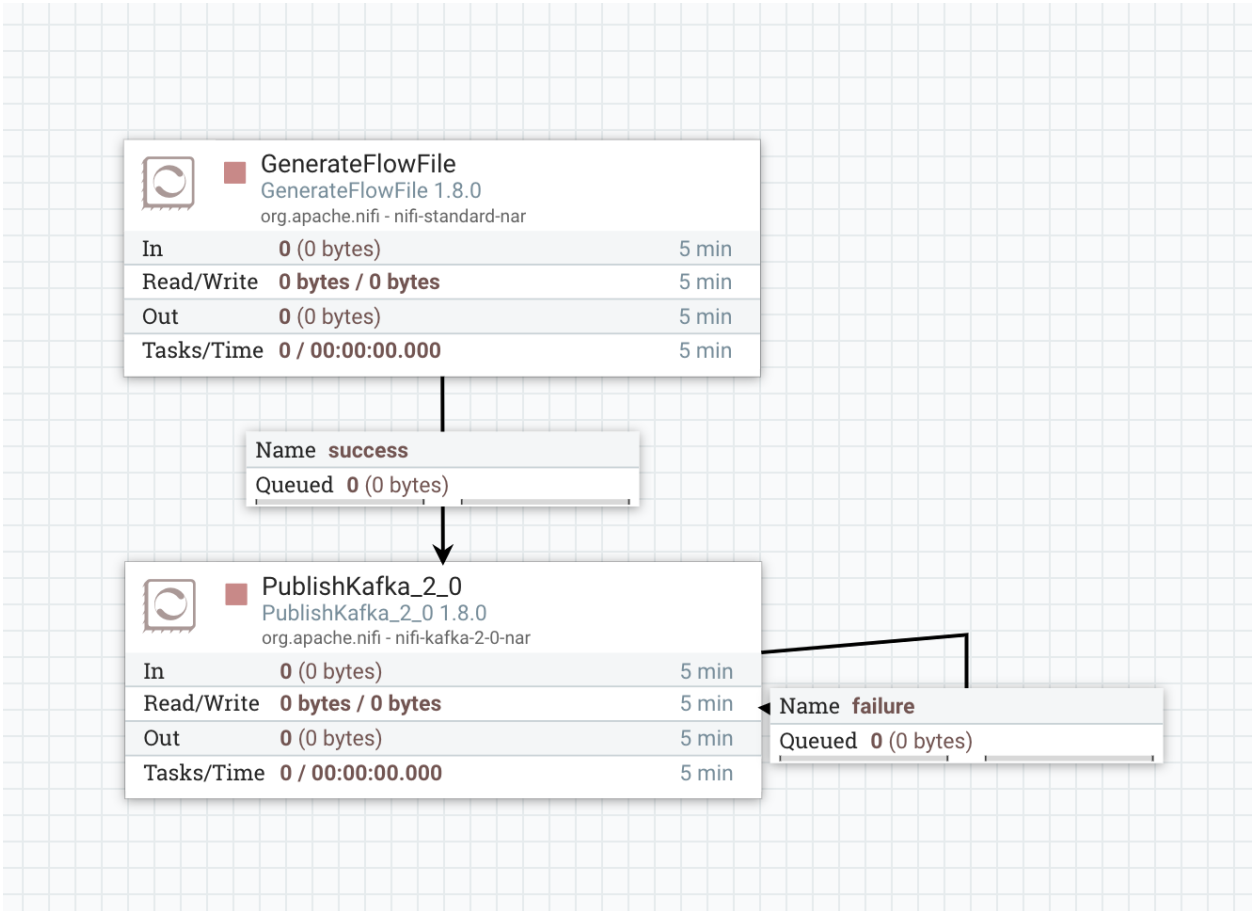


For sake of simplicity in this lab we have just added a self-loop in case of failure, but for real project is recommended to add an attribute to retry "x" time and after the number "x" tries have been failed then have additional steps for notifying the issue (send email, store in a file, log in APM apps, etc).

We can now terminate this flow by clicking on the PublishKafka_2_0 step, go to the **SETTINGS** tab and check the **success** box:

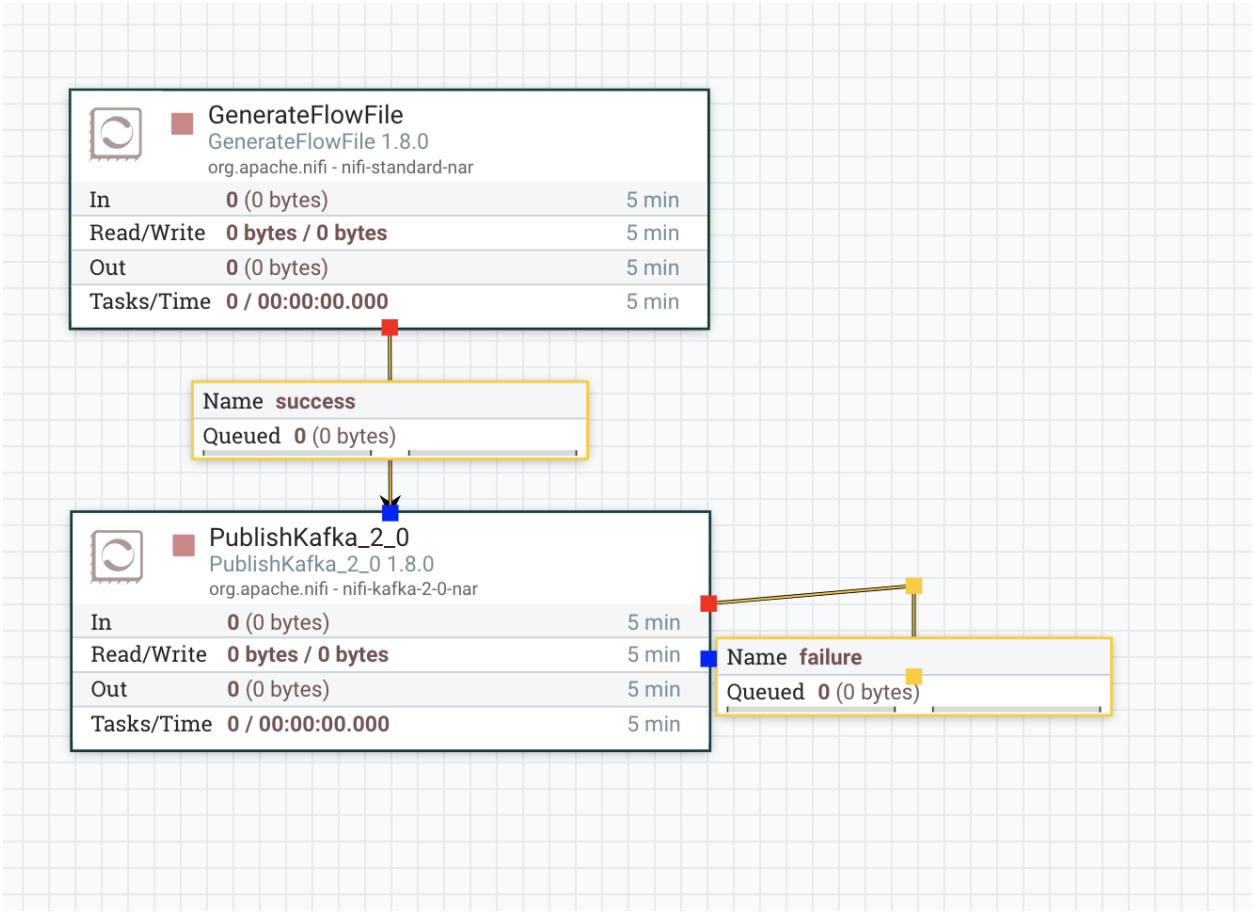


Then the warning icon on the top left corner of this component should be replaced by the "STOP" icon:

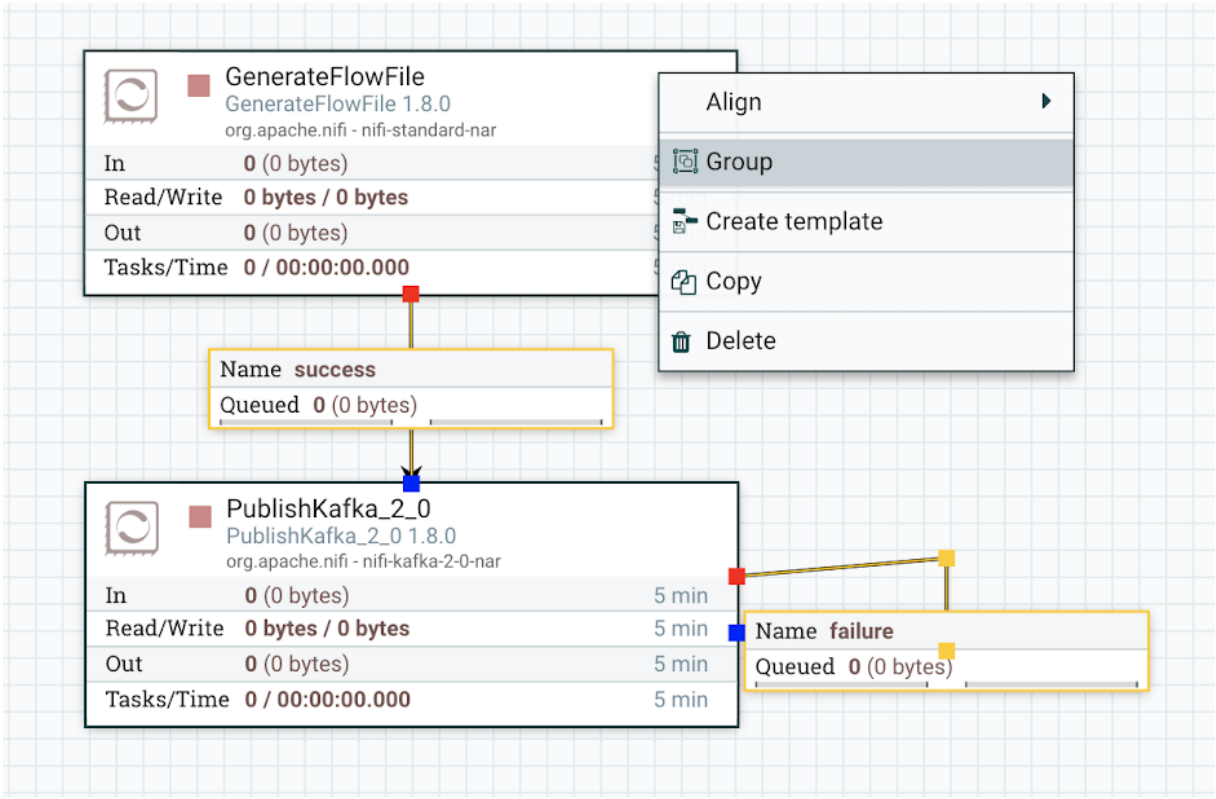


Now the flow for generating random sensors data reading is ready to go, but before run it let's create also the flow for ingesting the data into Kudu, as described in the next session.

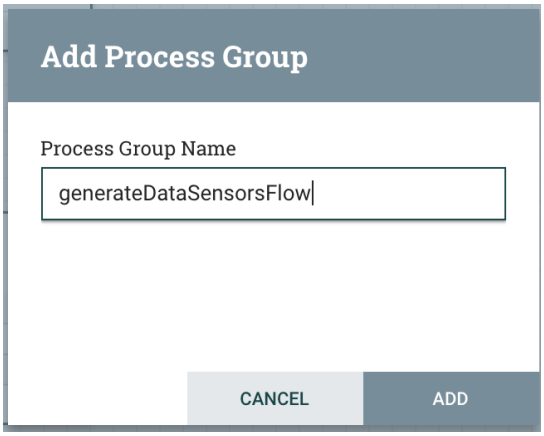
But before doing that, let's organize better what we have done so far.
With the mouse select a point in the canvas, press SHIFT and hold down the mouse and select your flow:



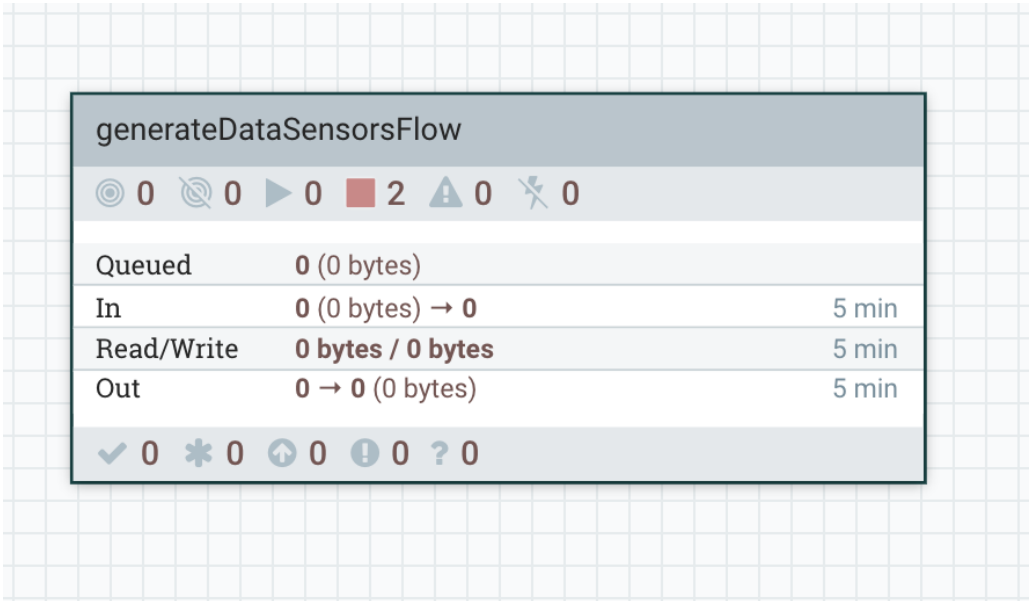
Then right click and click **Group**



Give it a name, in my case *generateDataSensorsFlow*:



And click **ADD**, we should get the following:



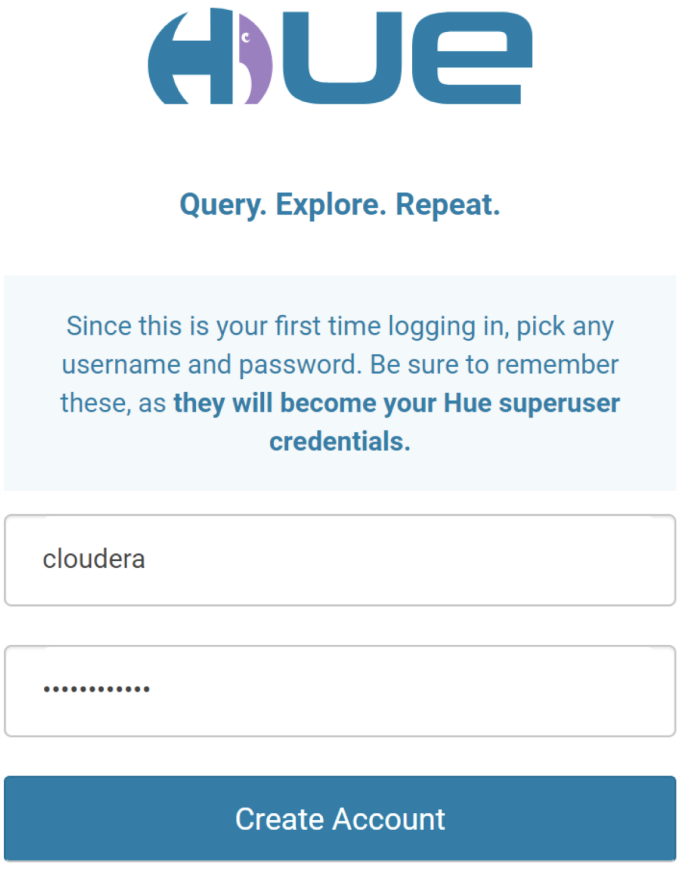
Now we are ready to move forward.

Lab 4: Define the Kudu tables for the Connected Thing time series data

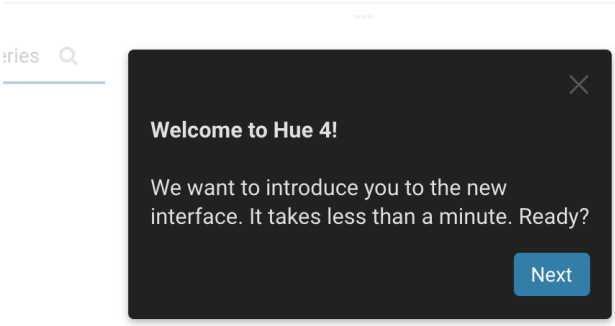
Us the following procedure to access the HUE interface in order to create the table that will be used to store the data from the remote Things.

Connect to Hue using the following url (don't forget to change the ip address):
http://40.76.204.159:8888

If it's the first time to access HUE, then you will need to create an account (I'd suggest you use **cloudera** and **Cloudera_123**)



Clouse with pop-up window:

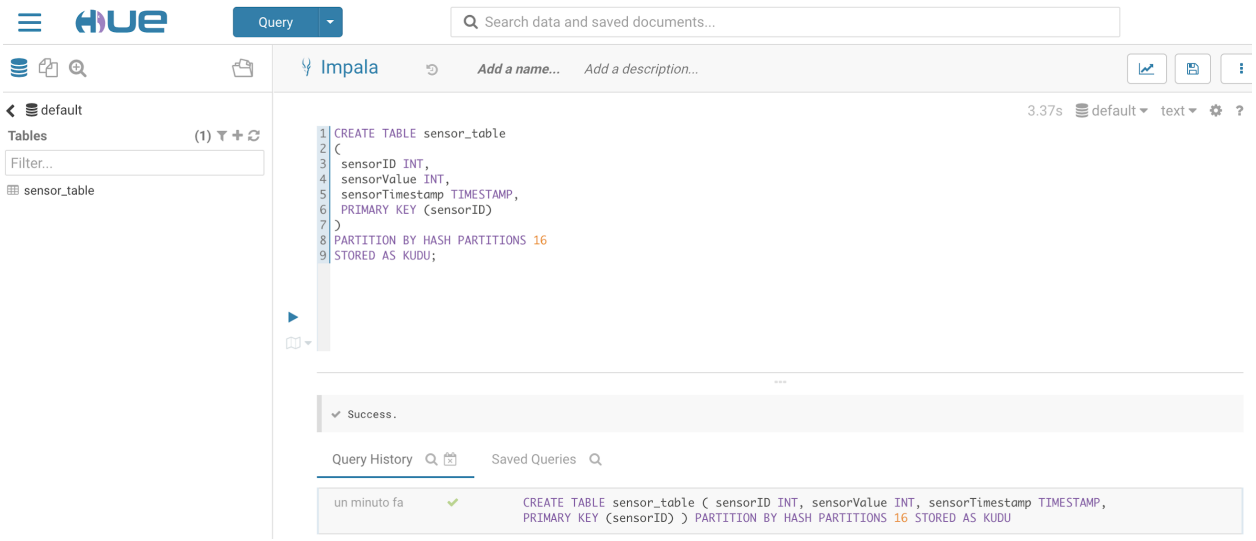


Within the Impala Query Editor type the following SQL statement:

```
CREATE TABLE sensor_table
(
 sensorID INT,
 sensorValue INT,
 sensorTimestamp TIMESTAMP,
 PRIMARY KEY (sensorID)
)
PARTITION BY HASH PARTITIONS 16
STORED AS KUDU;
```

The above statement is going to create a table, called sensor_table, with 3 fields, the sensorID as primary key, and 16 hash partitions stored as Kudu.

Click the run icon 

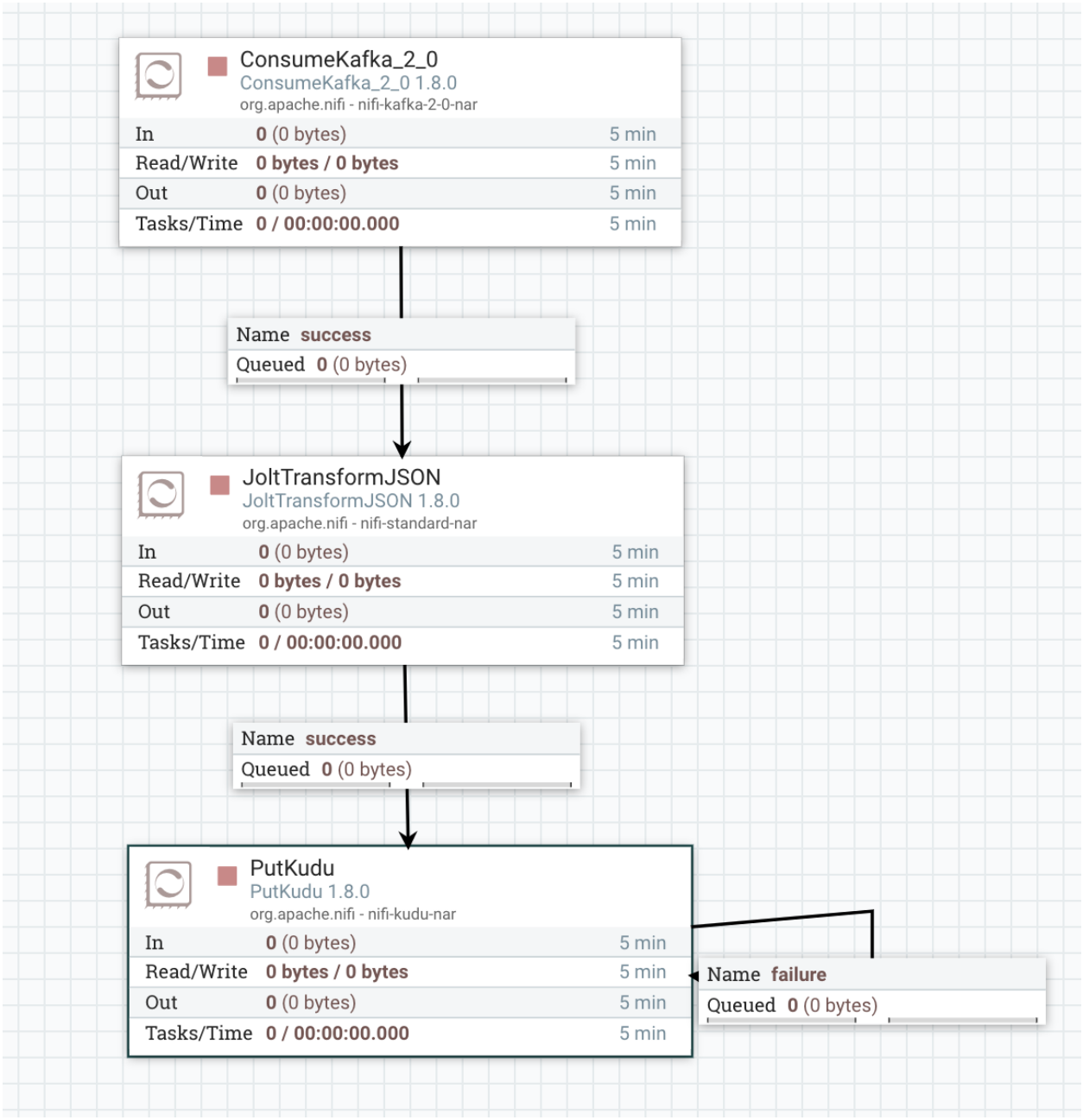


Lab 5: Create the real-time ingest pipeline

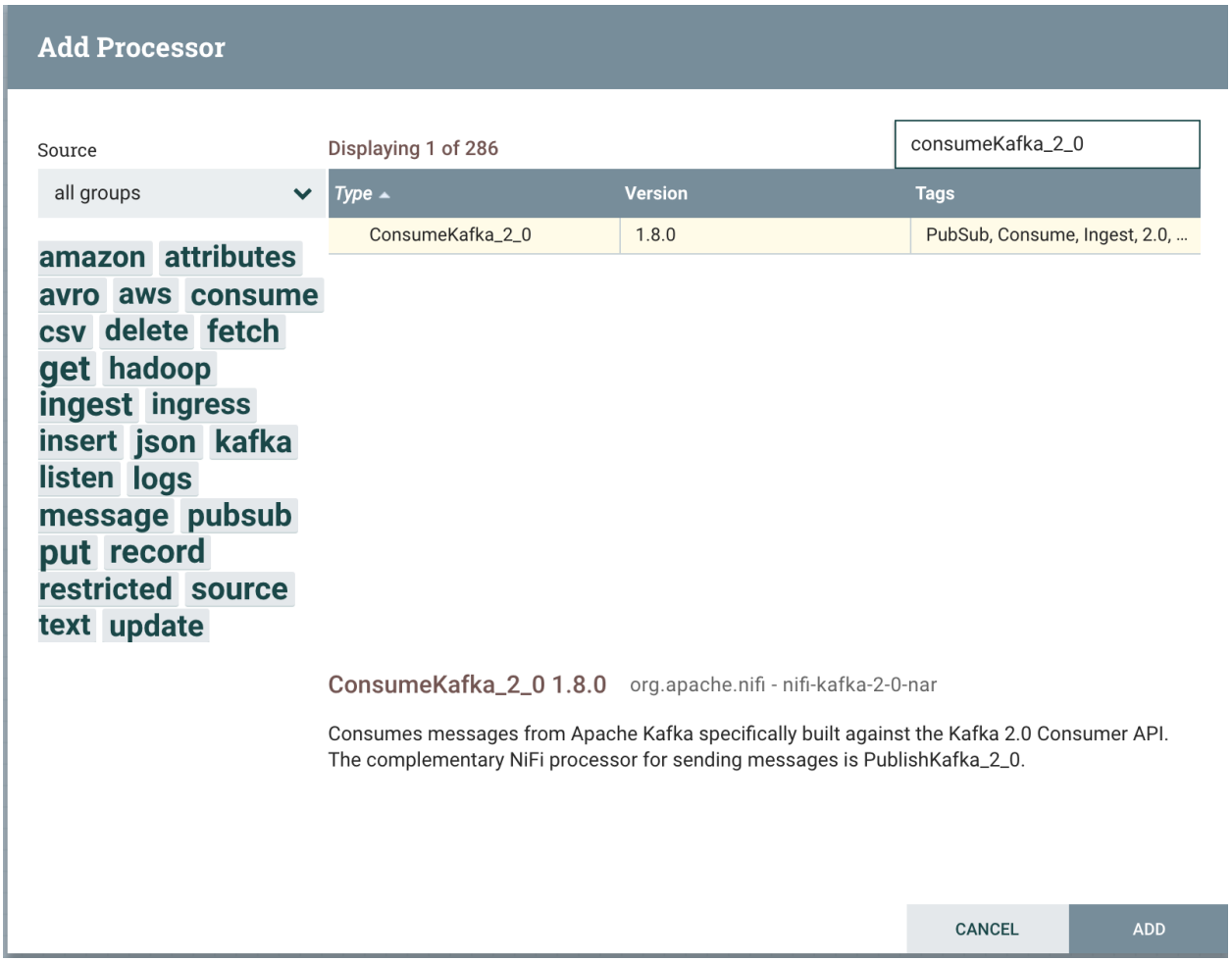
Now that we have a way to simulate the time series readings from 10,000 sensors and the kudu table to store the data, we can proceed further and create the real-time ingestion pipeline that will take the readings from the remote Things and update the Kudu table.

ingestDataSensors flow

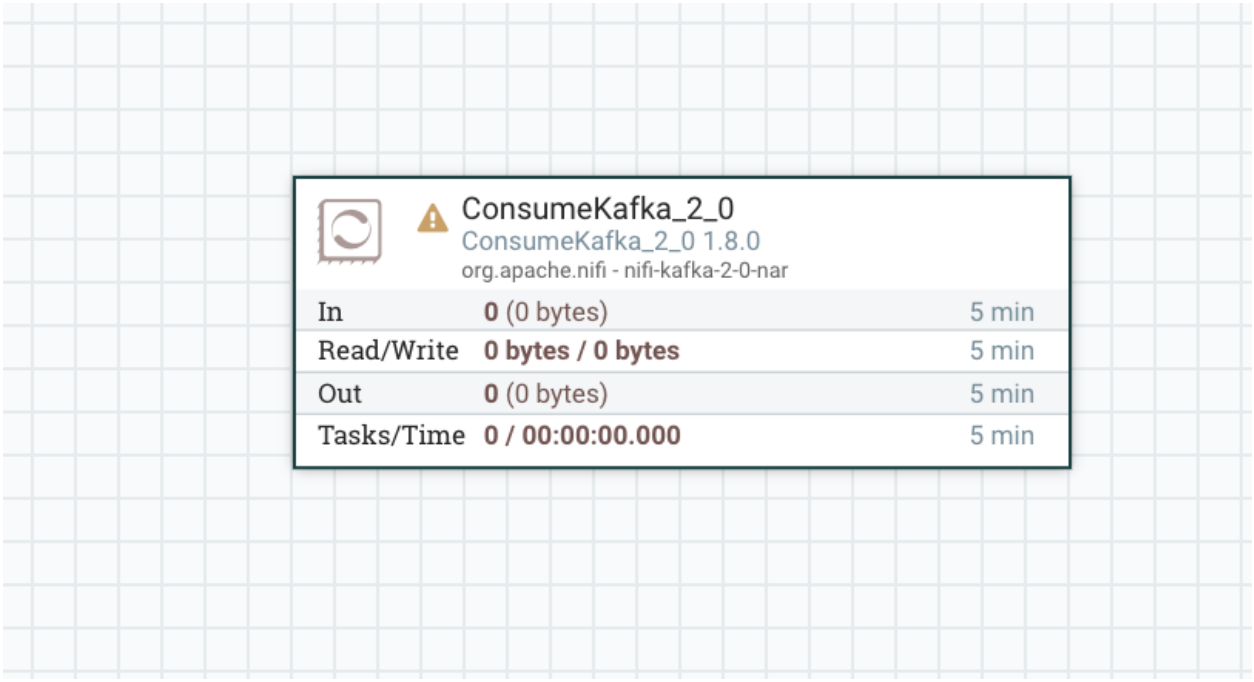
This flow is going to look like the following picture



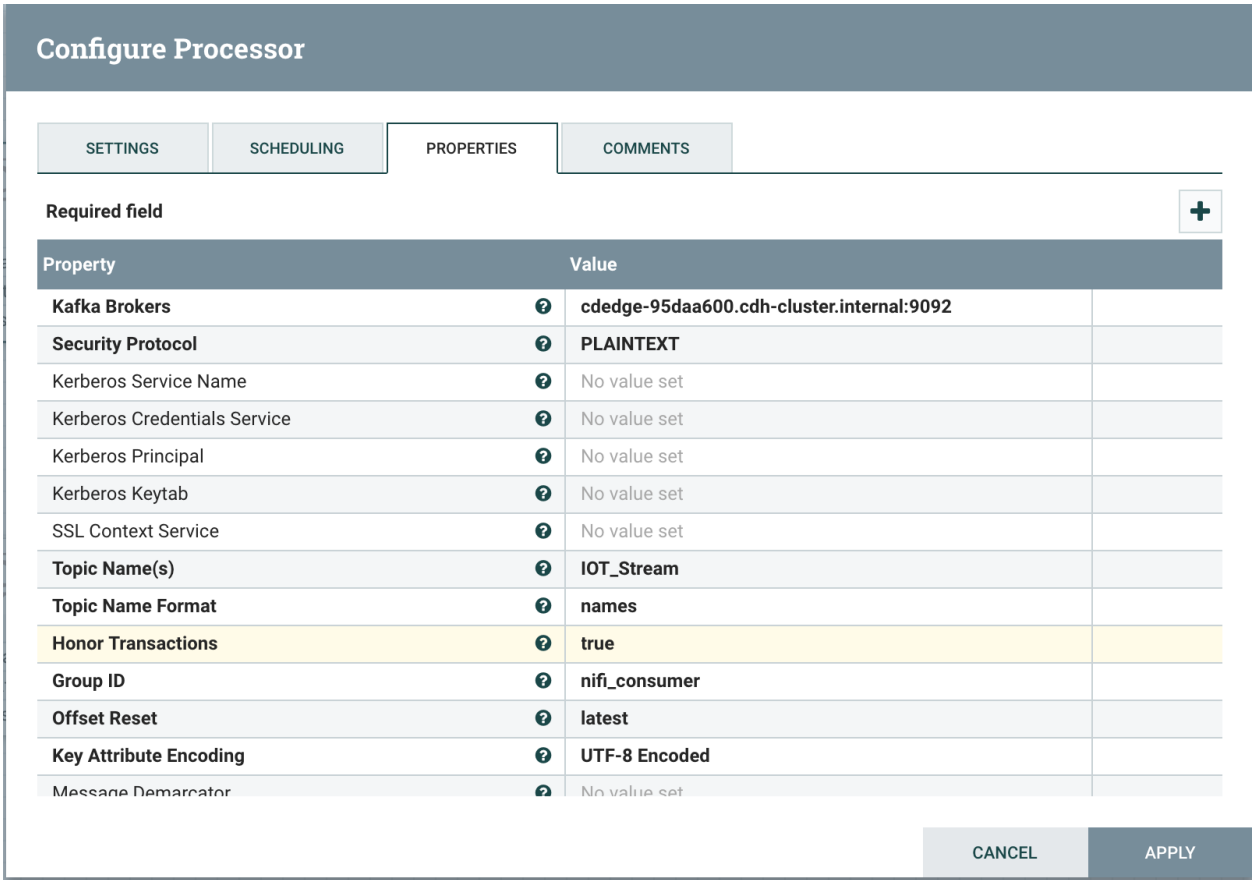
Now you should be already familiar on how to add a new processor to the canvas, then add the **ConsumeKafka_2_0** processor:



Click **ADD**



Double click on the **ConsumeKafka_2_0** and go to the **PROPERTIES** tab



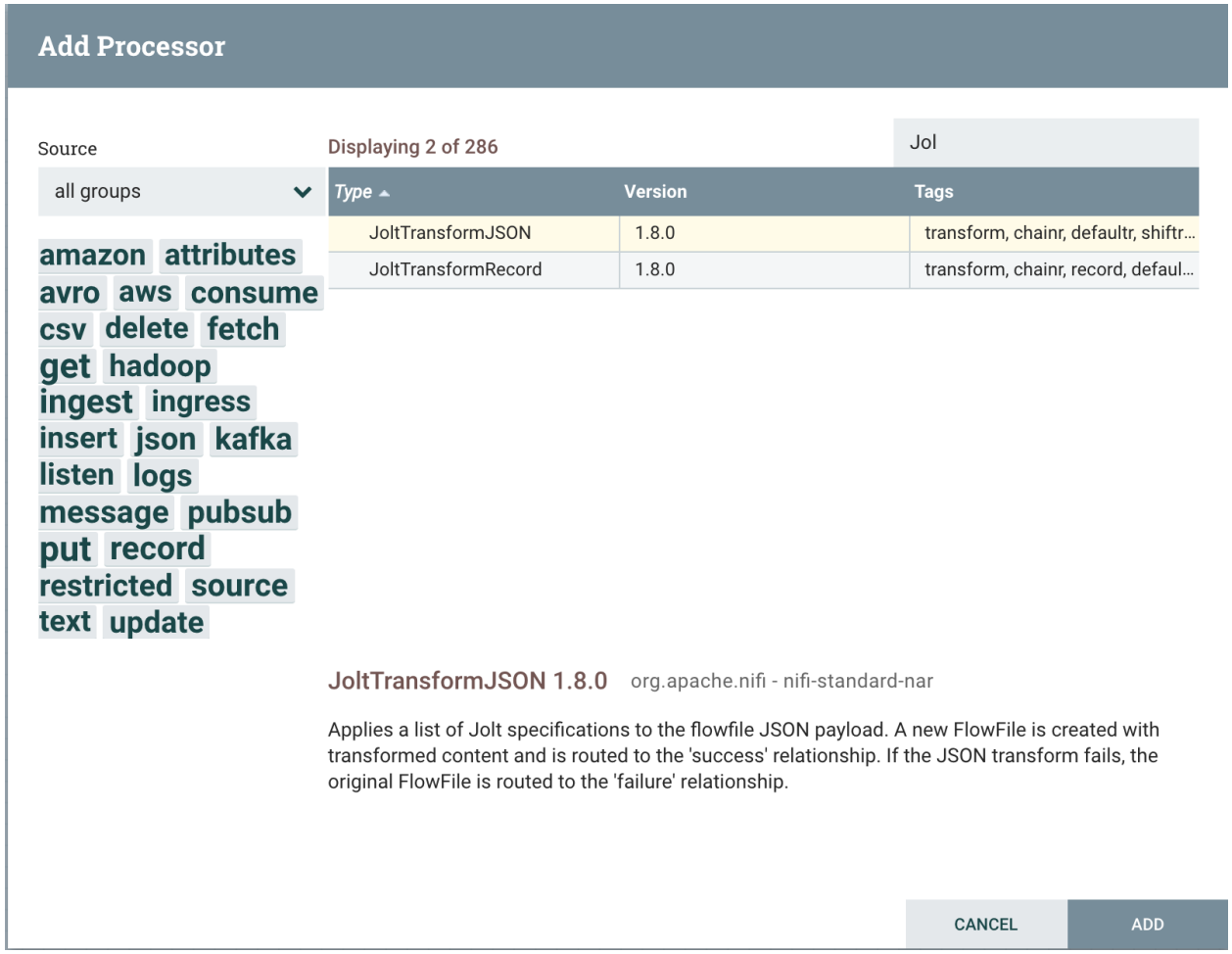
And add the value for the following properties:

- **Kafka Brokers**: <your-private-ip>:9092 (in my example: 10.0.0.4:9092)
- **Topic Name(s)**: IOT_Stream
- **Group ID**: nifi_consumer

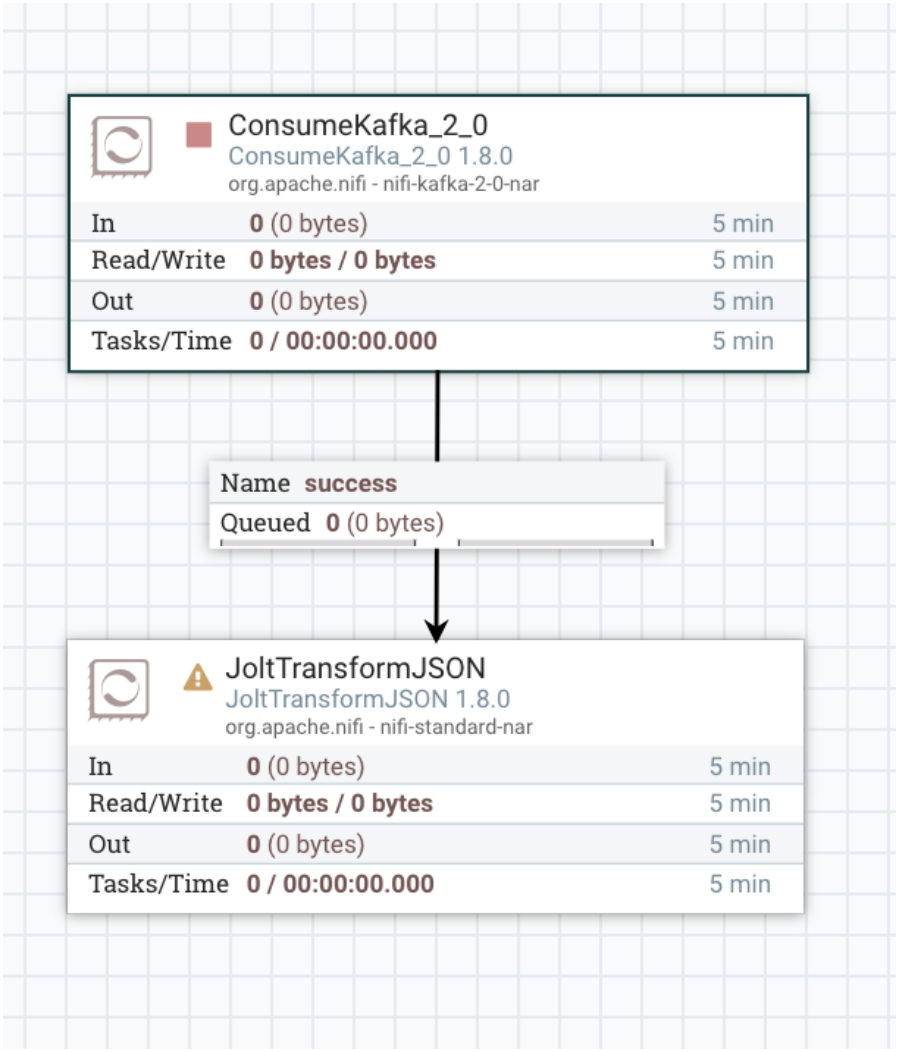
Once done, click **APPLY**

Now we need to transform the message received by Kafka consumer in order to adapt it to the kudu schema table. In particular we need to transform all the fields in lower letter.

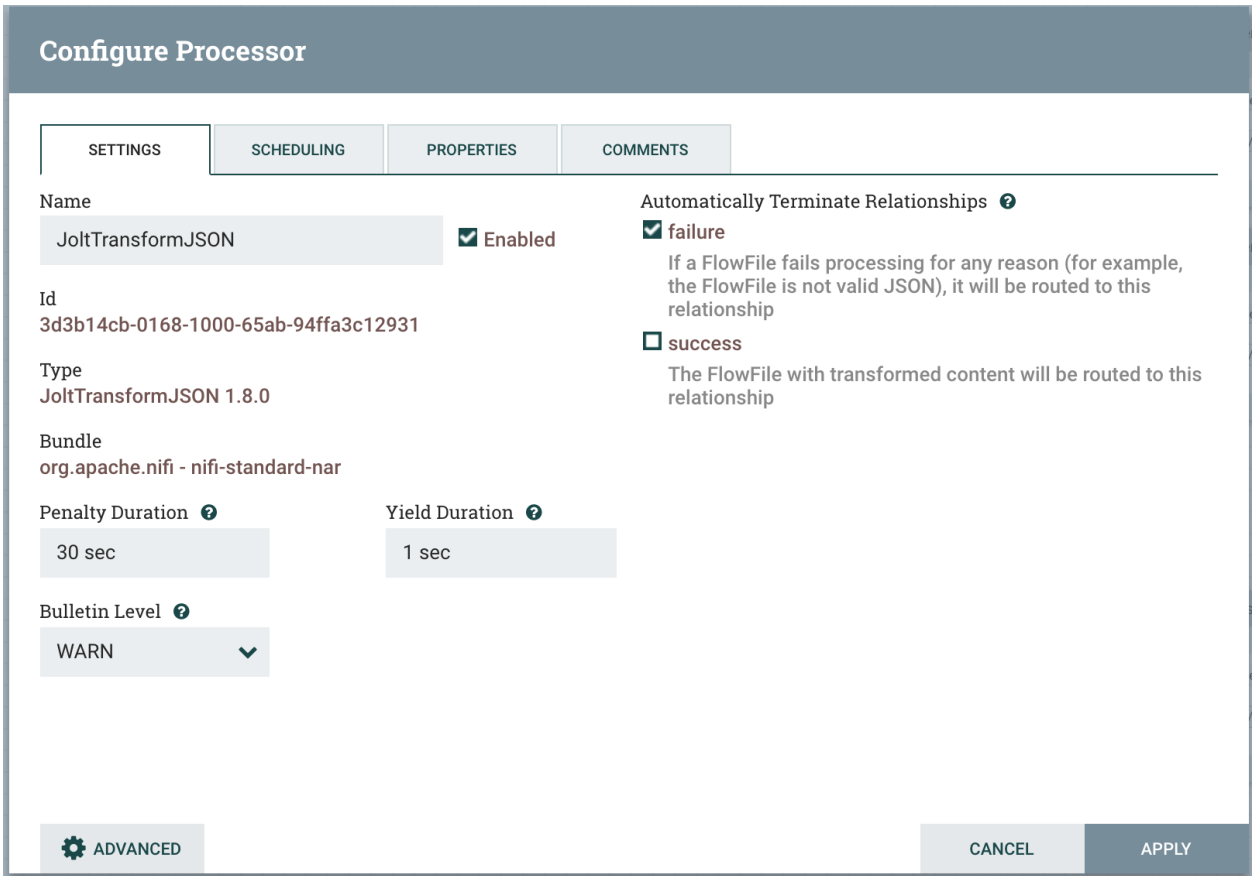
Add a **JoltTransformJSON** to the canvas:



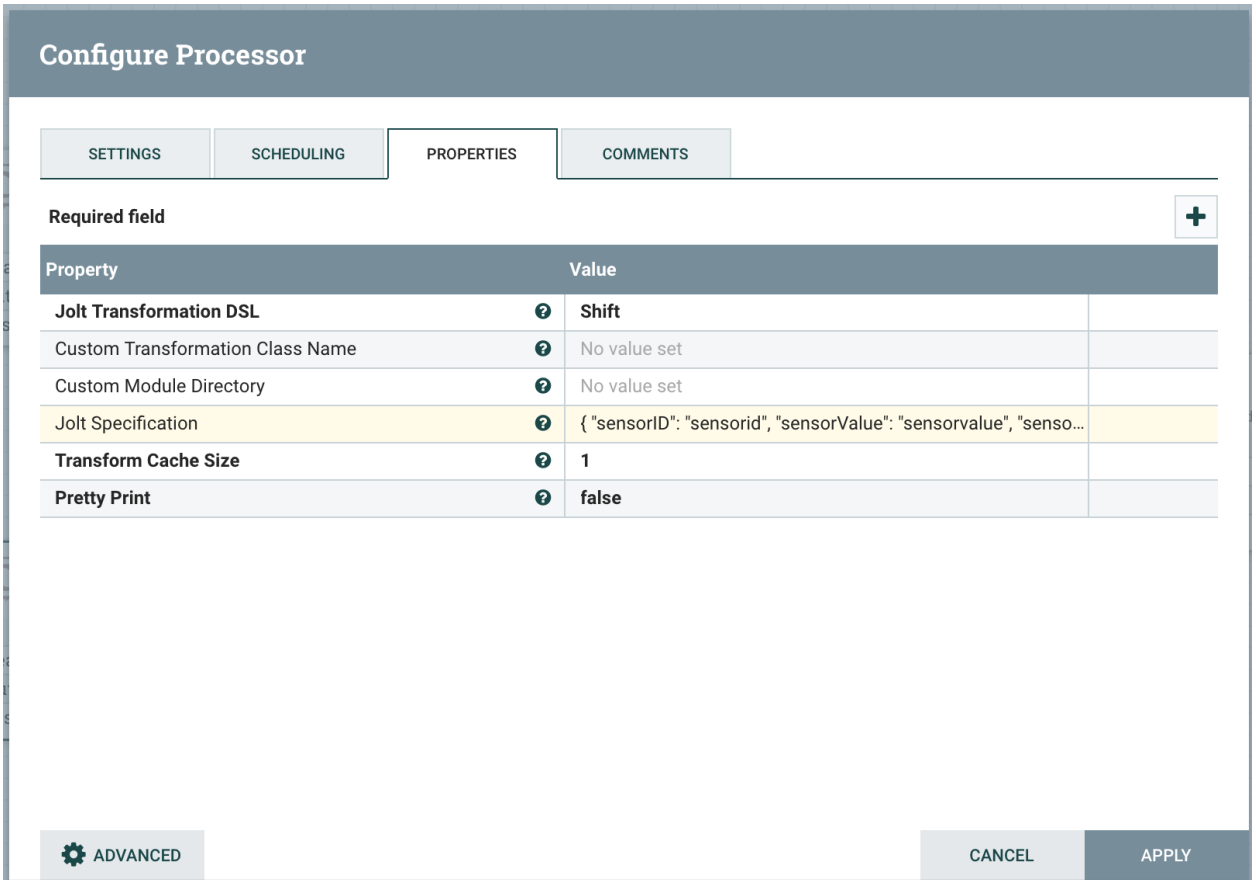
Click **ADD** and connect the kafka consumer to this component:



Connect the kafka consumer to the transform step, now you should know how to do it.
Double click on the **JoltTransformJSON** component and go to the **SETTINGS** tab



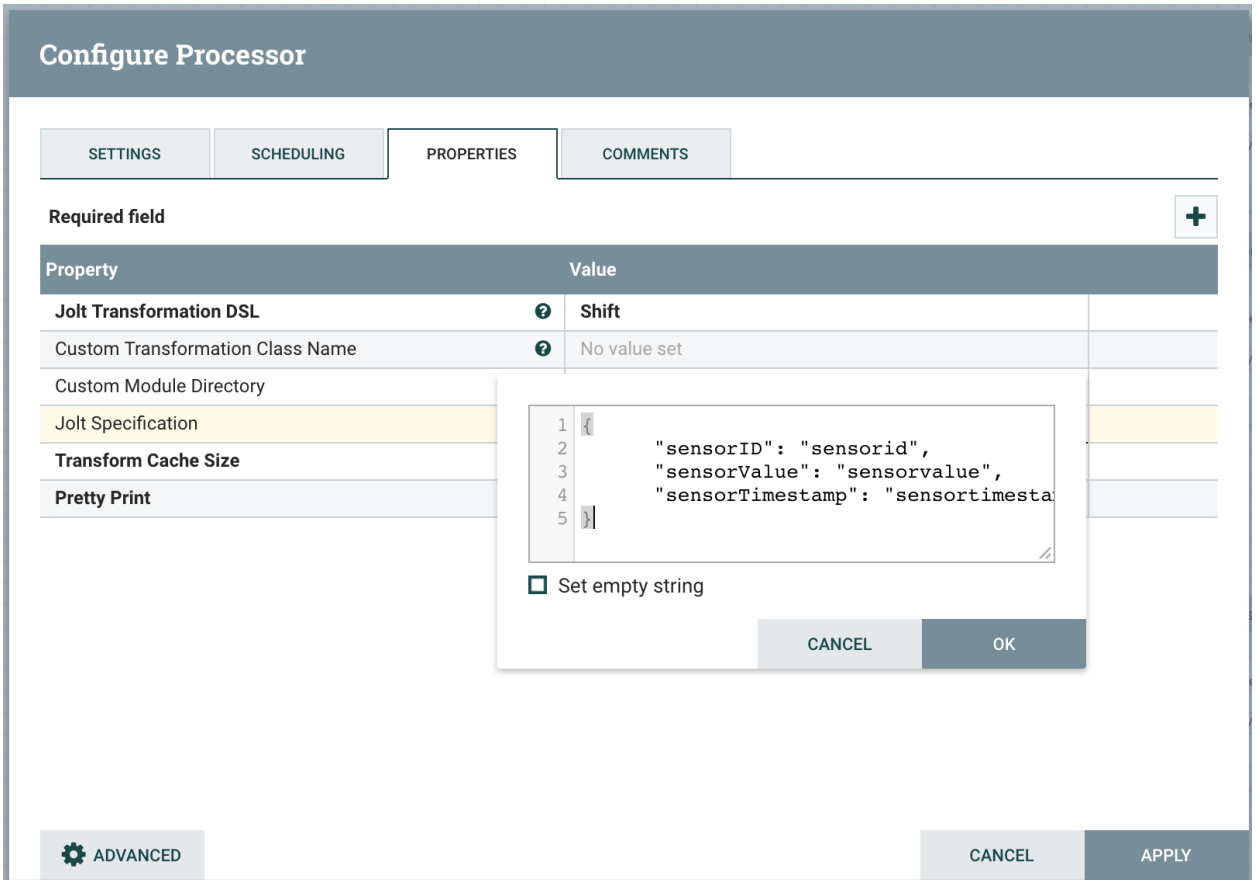
Check the **failure** box and go to the **PROPERTIES** tab



Add the following values:

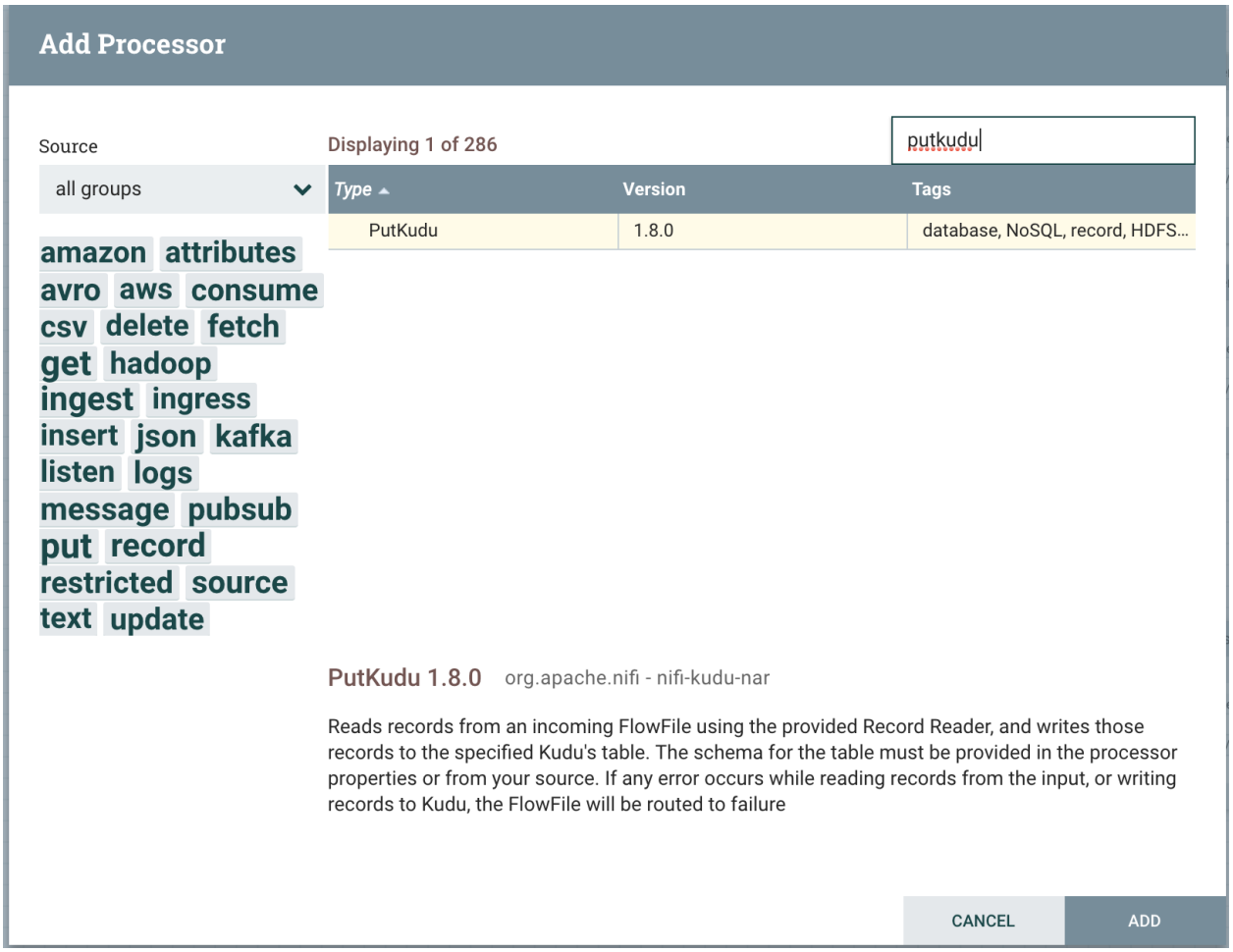
- **Jolt Transformation DSL**: Shift
- **Jolt Specification**:

```
{
    "sensorID": "sensorid",
    "sensorValue": "sensorvalue",
    "sensorTimestamp": "sensortimestamp"
}
```

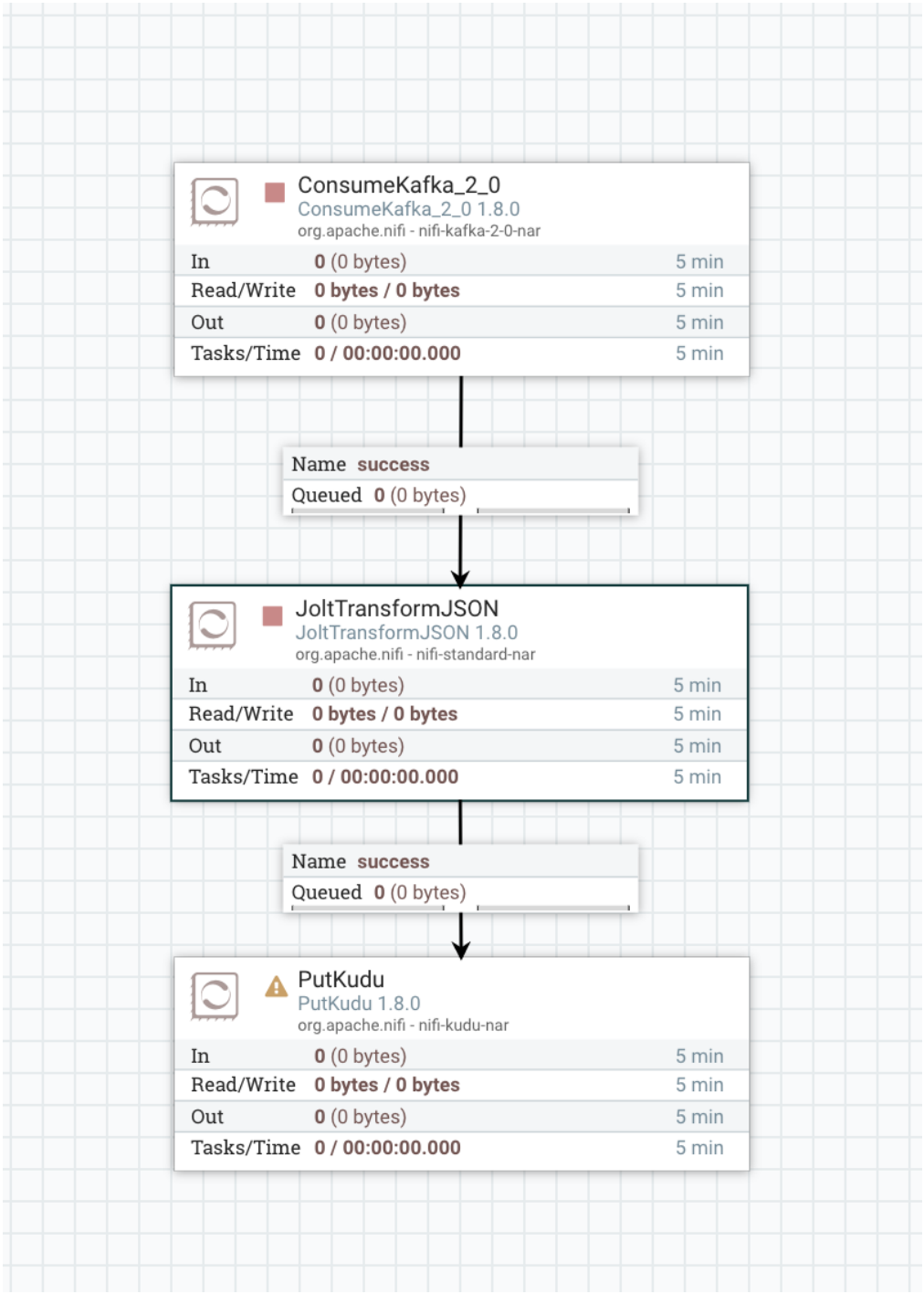


Then click **OK** and **APPLY**.

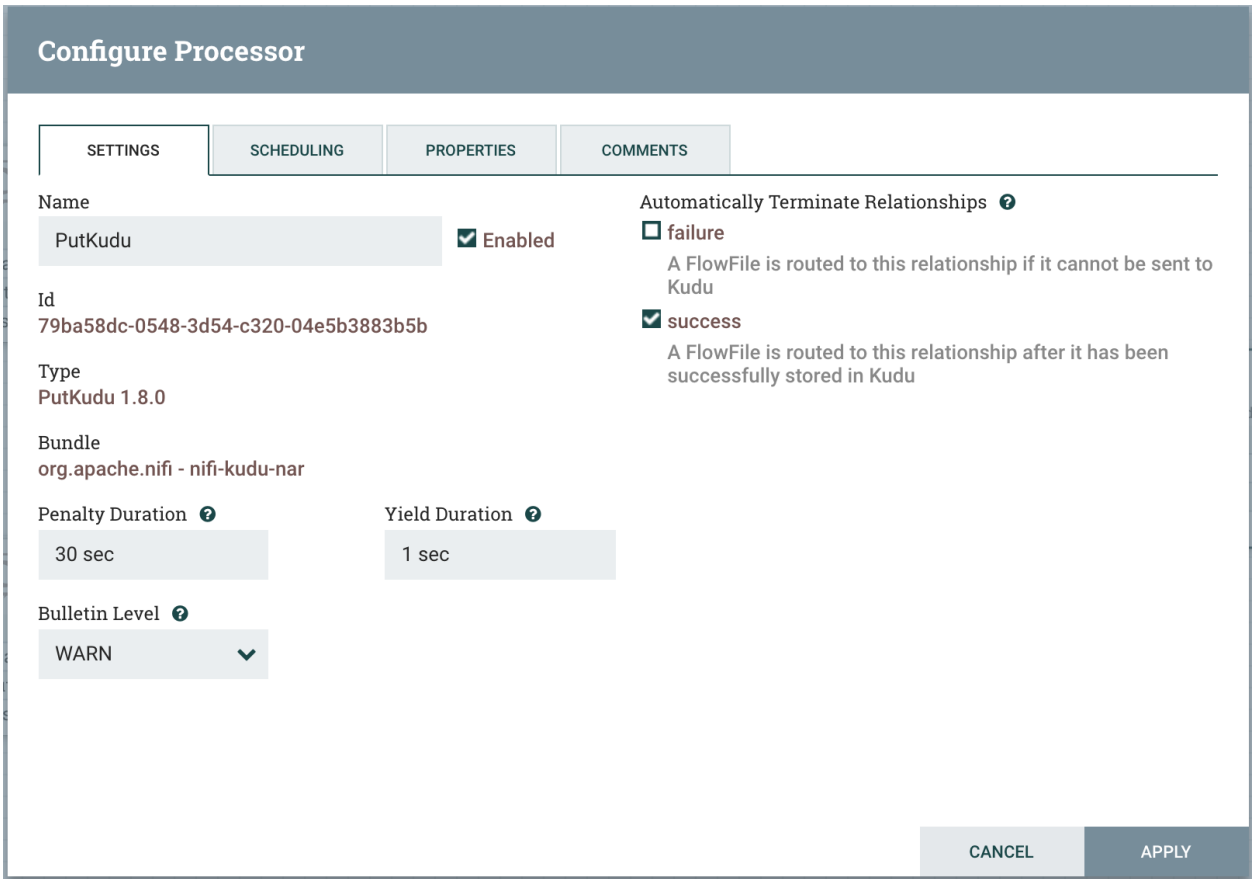
Now add a **PutKudu** processor to the canvas



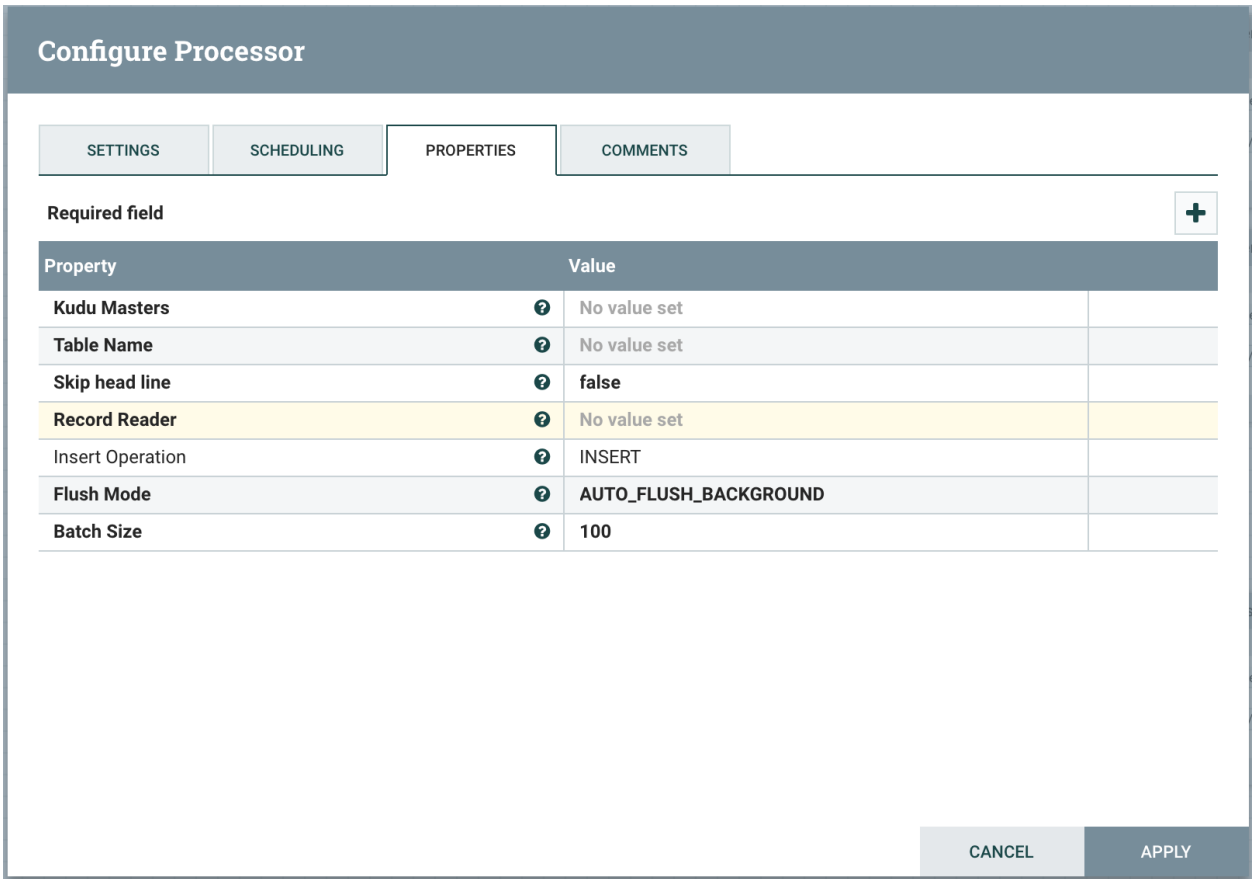
Click **ADD** and connect to the **JoltTransformJSON**



Double click on PutKudu step and go to the **SETTINGS** tab



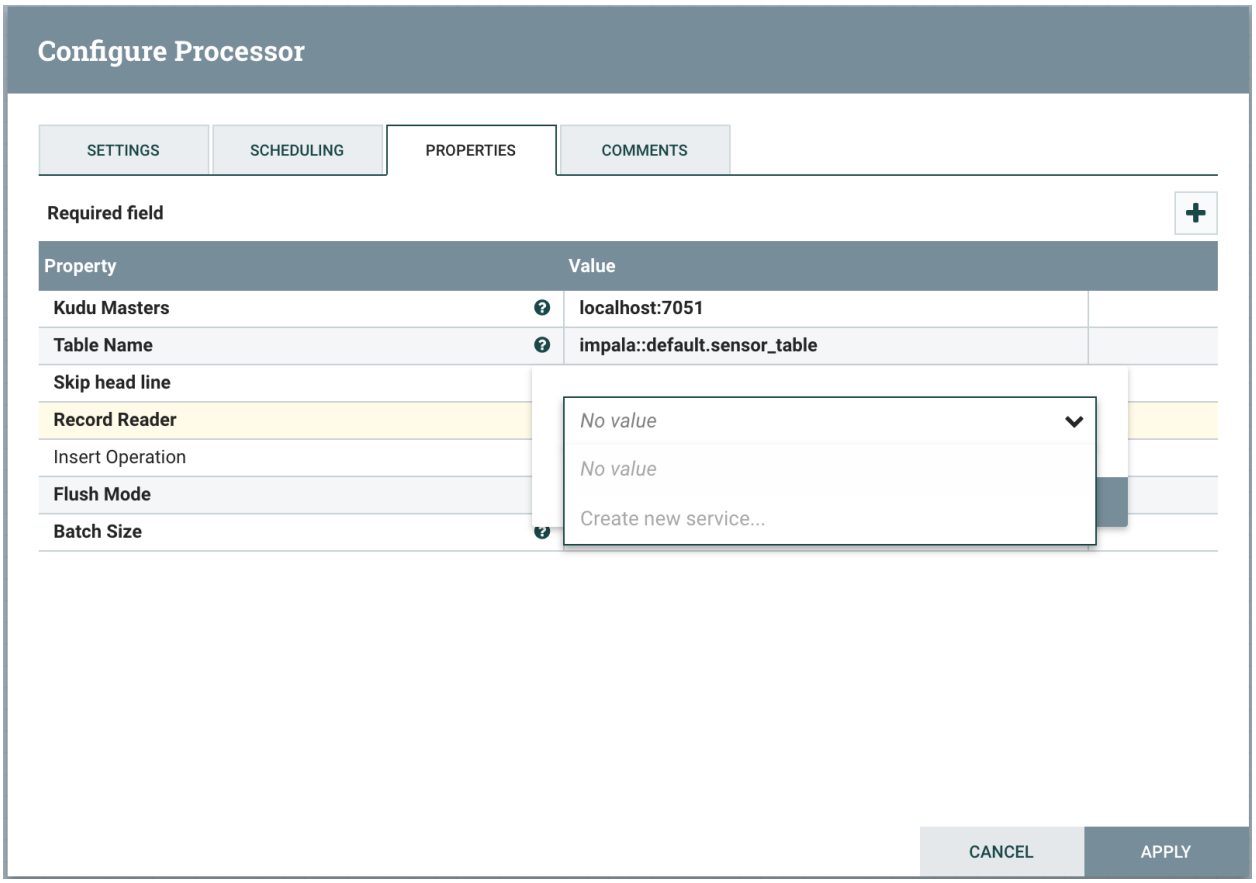
Check the **success** box and go to the **PROPERTIES** tab



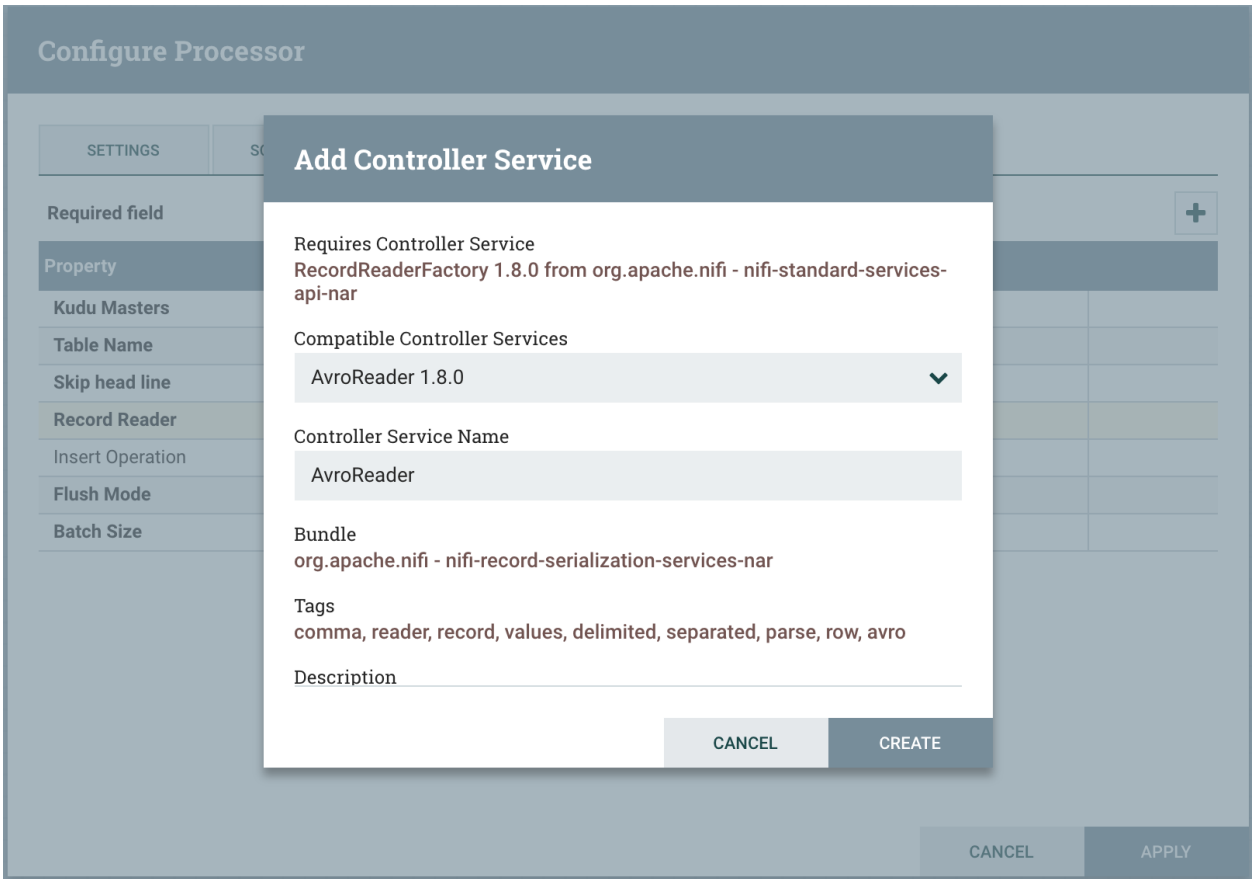
Set:

- **Kudu Masters**: *<your-private-ip>:7051* (in my case 10.0.0.4:7051)
- **Table Name**: impala::default.sensor_table
- **Insert Operation**: UPSERT

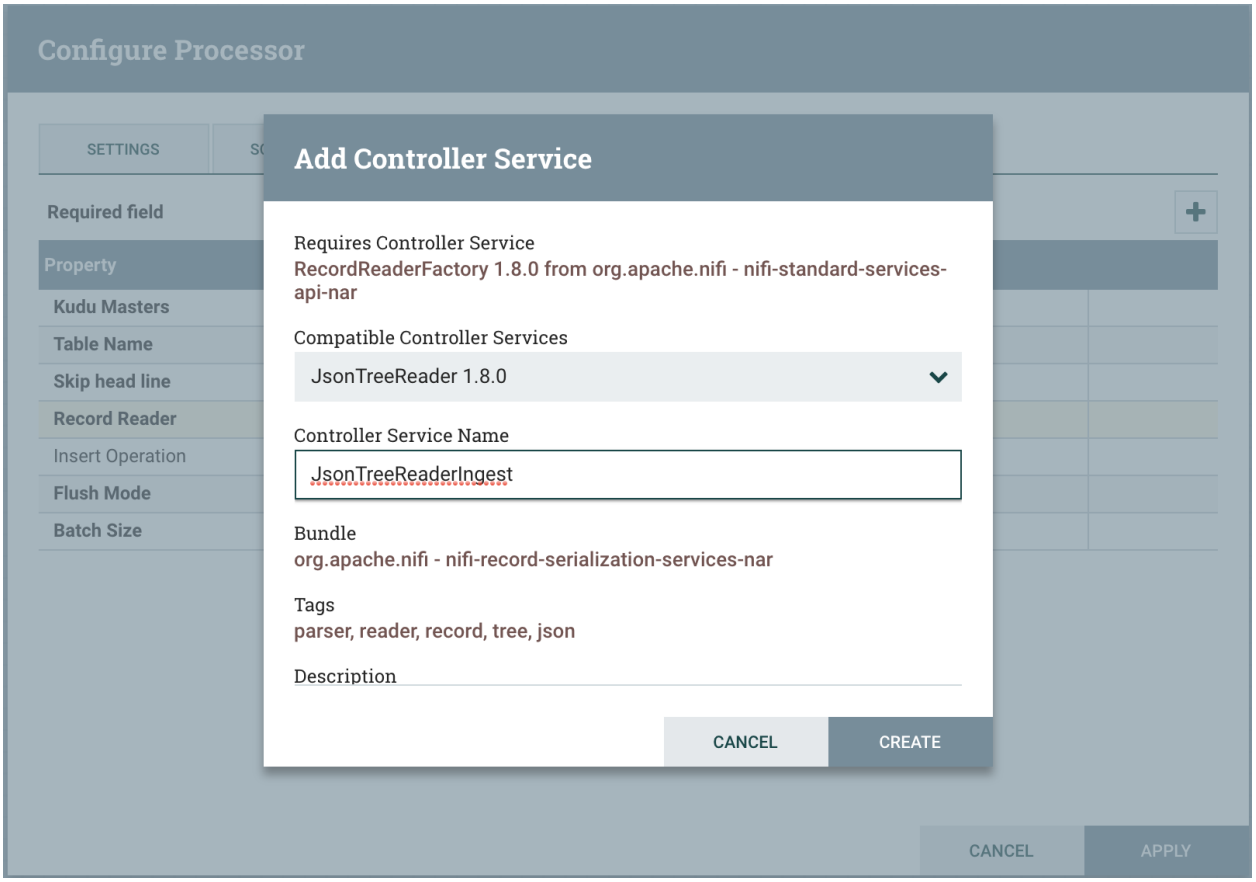
Then for the **Record Reader** ...



click on **Create new services...**

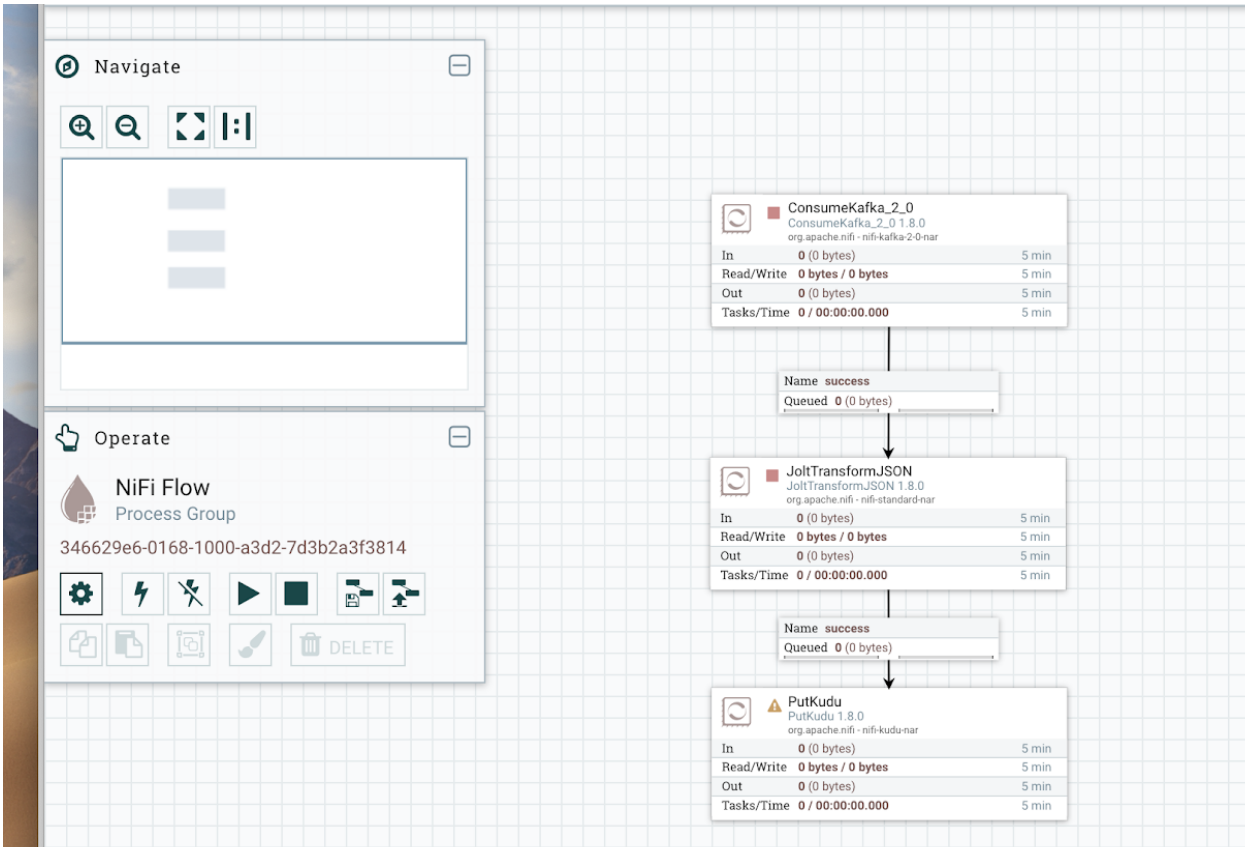


In the **Compatible Controller Services** select **JsonTreeReader** and give it a name (JsonTreeReaderIngest)

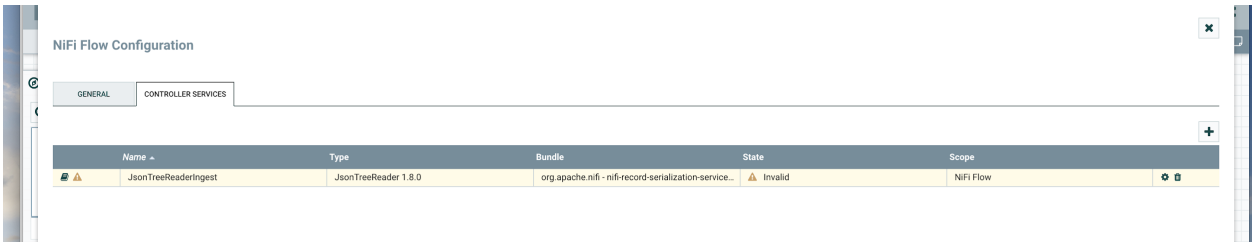


Then click **CREATE**.

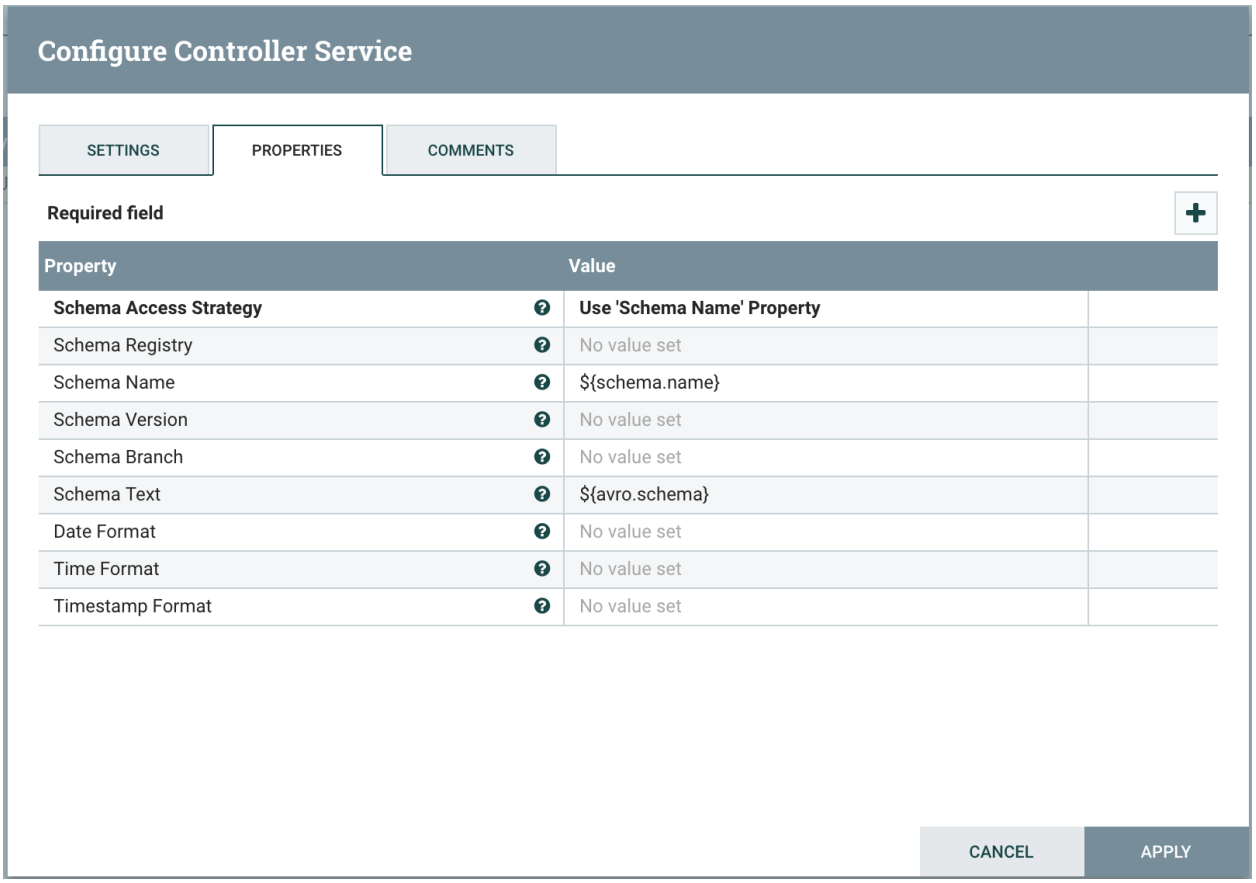
Then click **APPLY**.
Deselect the **PutKudu** processor by clicking on the canvas, and then click the **Configuration** icon in the **Operate** box



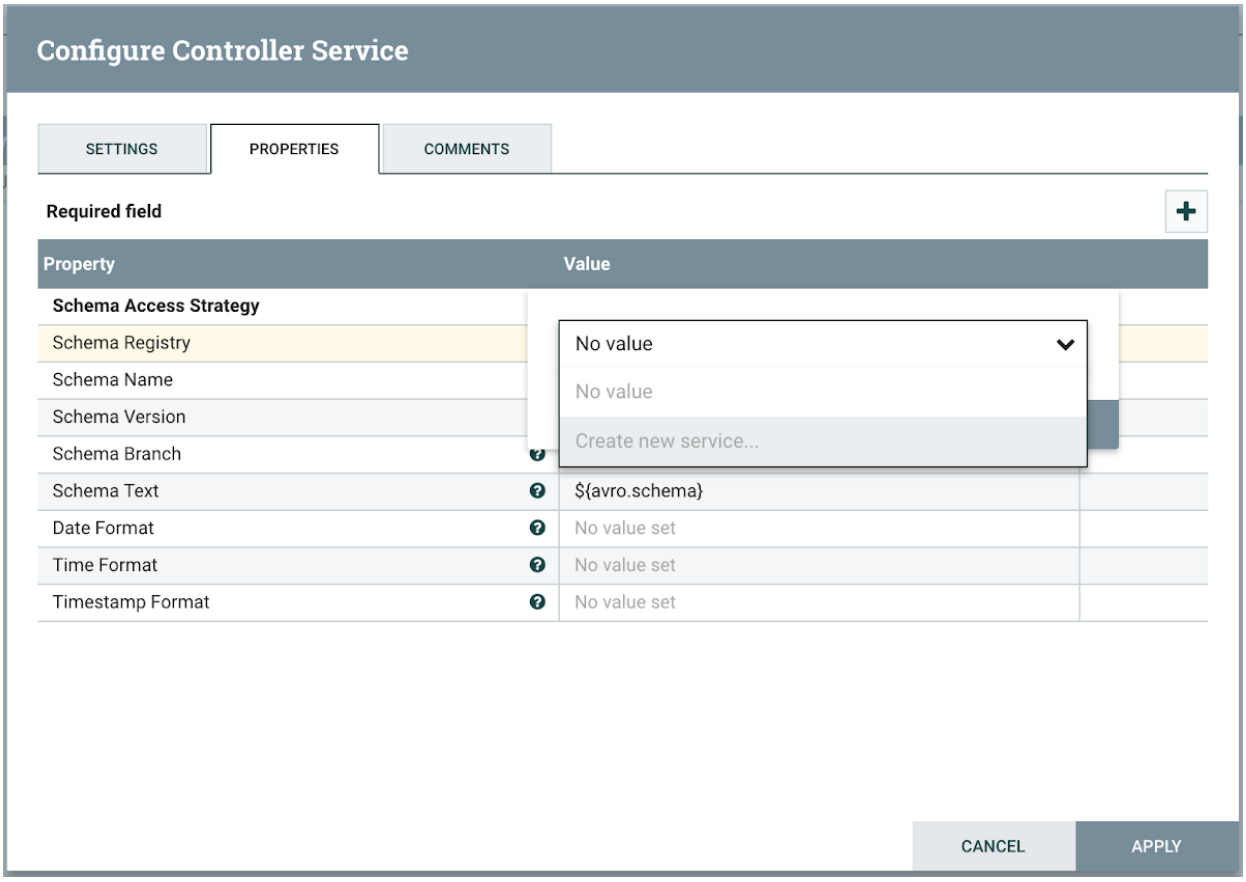
Then go to the **CONTROLLER SERVICES** tab



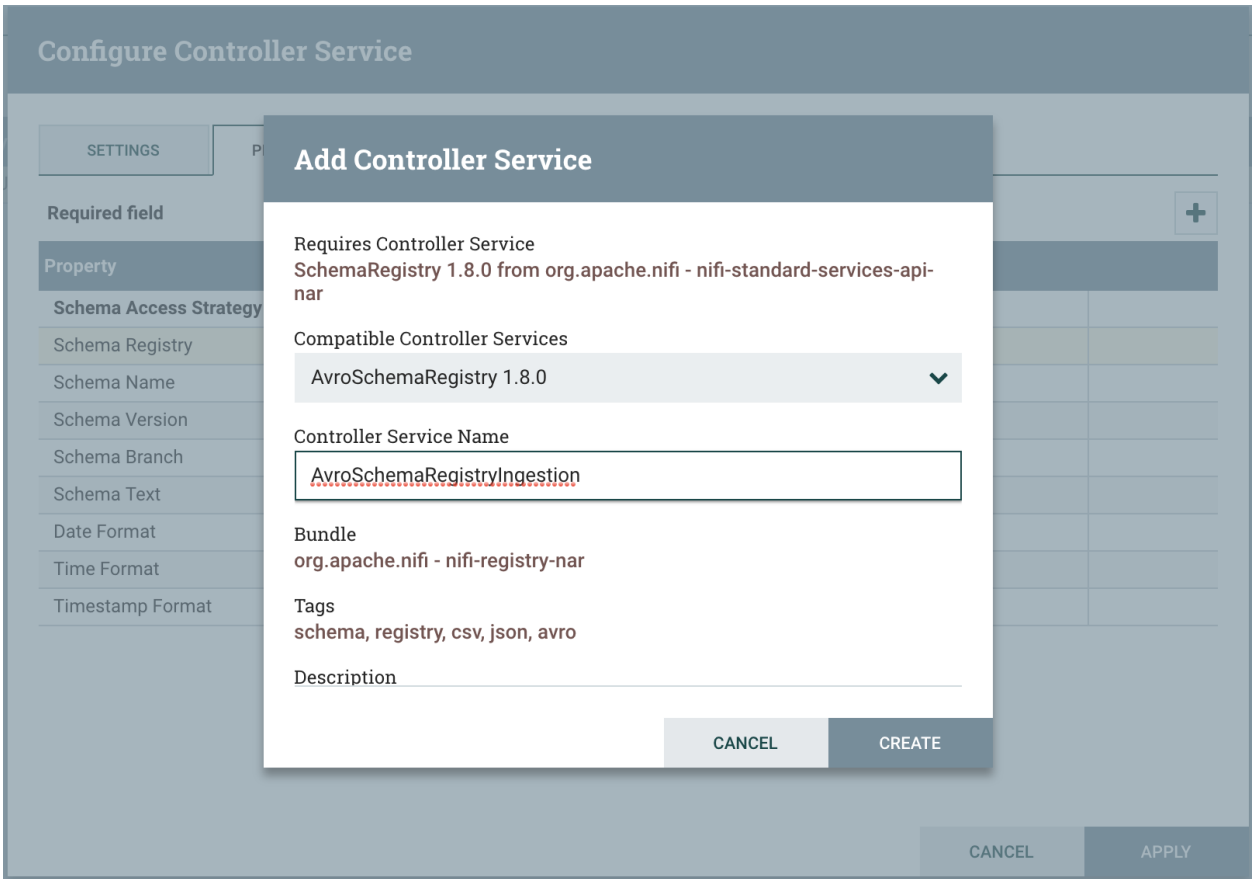
Click on the **configure** icon on the right end of the **JsonTreeReaderIngestion** row and go to the **PROPERTIES** tab



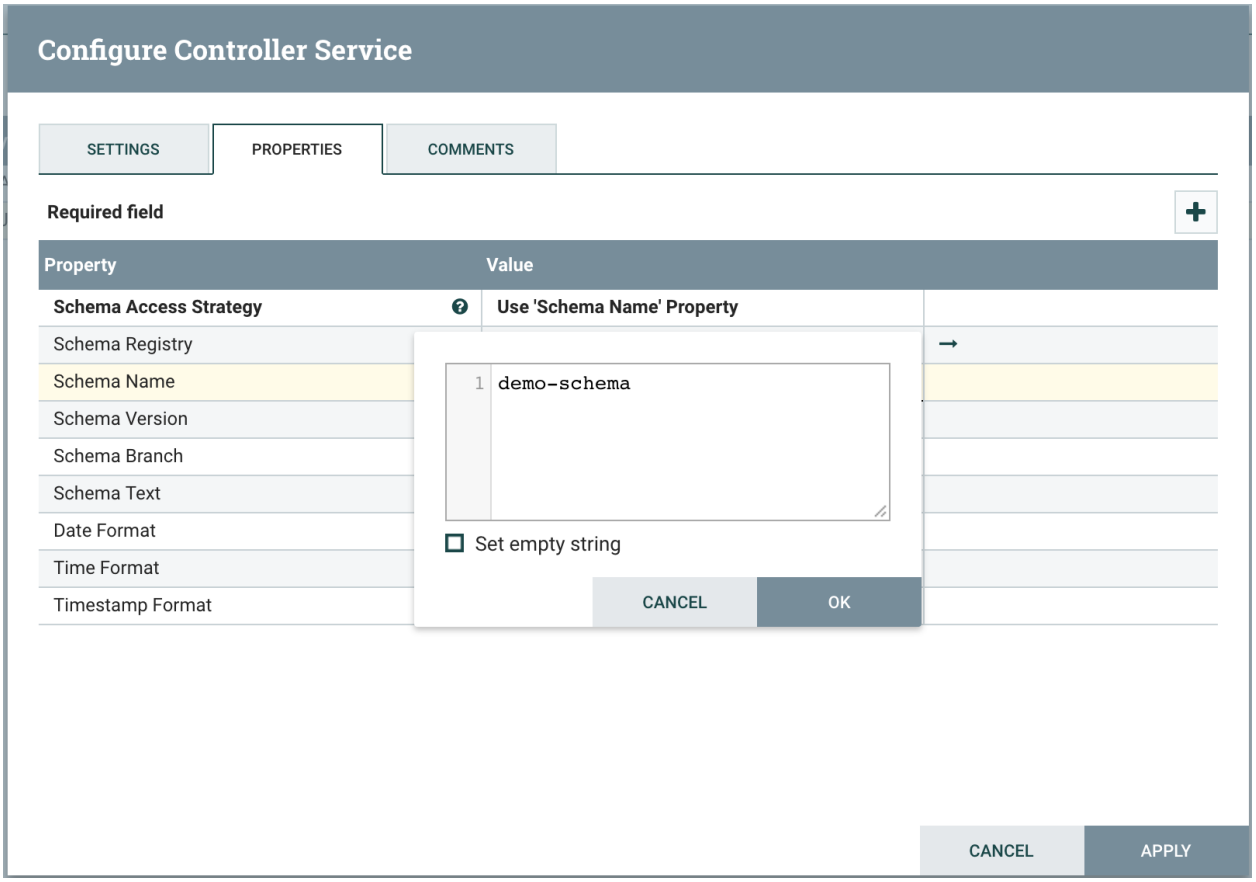
In the **Schema Registry** field click to the value field, click **Create new service**...



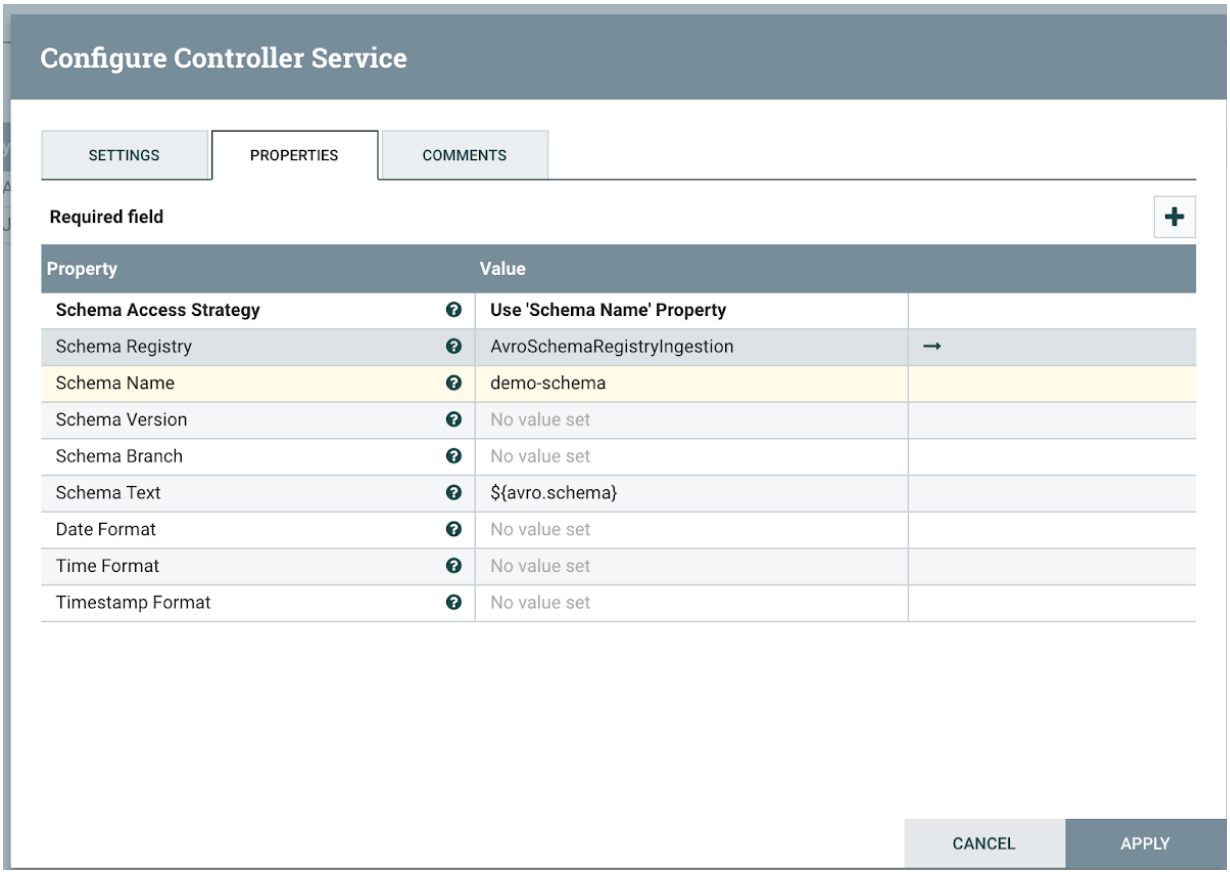
Select **AvroSchemaRegistry 1.8.0** and give it a name like *AvroSchemaRegistryIngestion*.



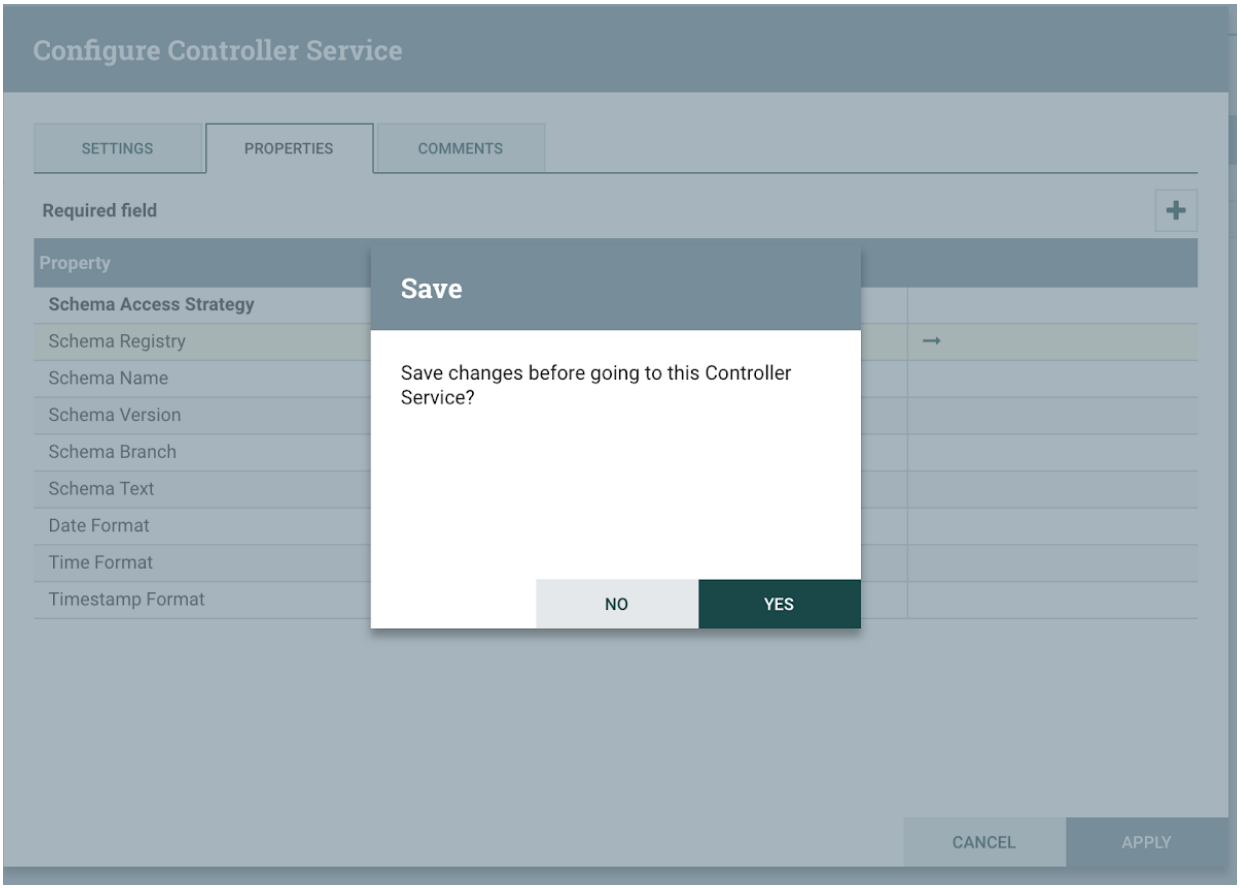
Then click **CREATE**.
In the **Schema Name** field set *demo-schema*



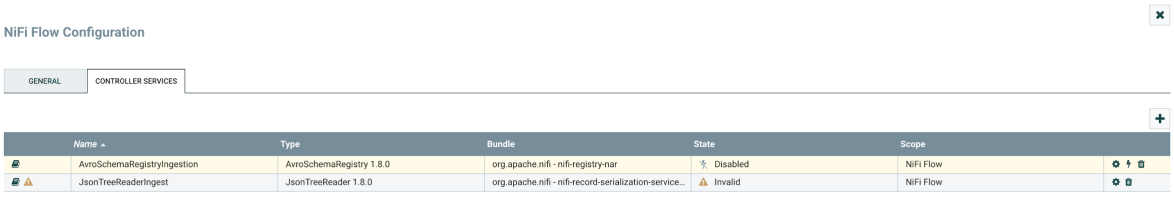
Click **OK**
Now click on the arrow icon in the **Schema Registry** row



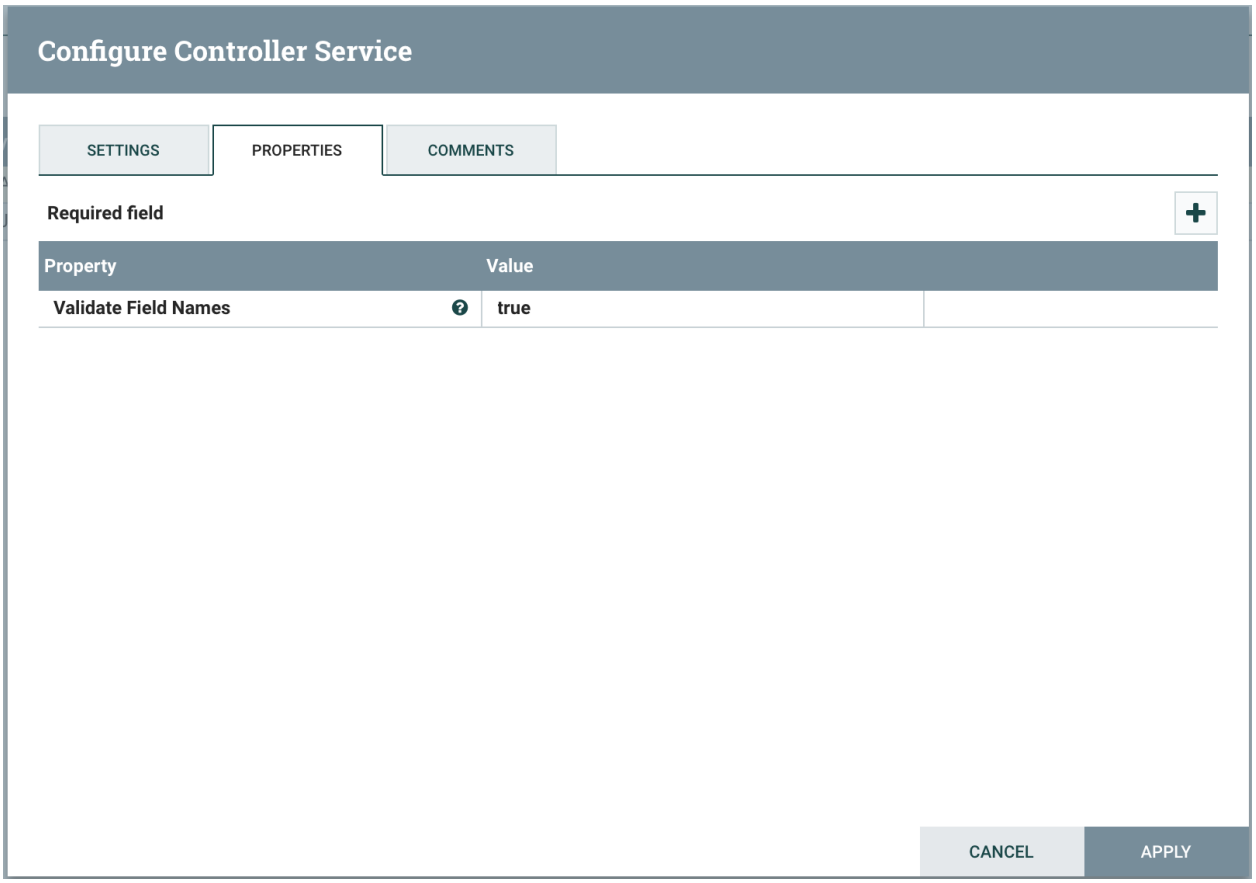
Save the change



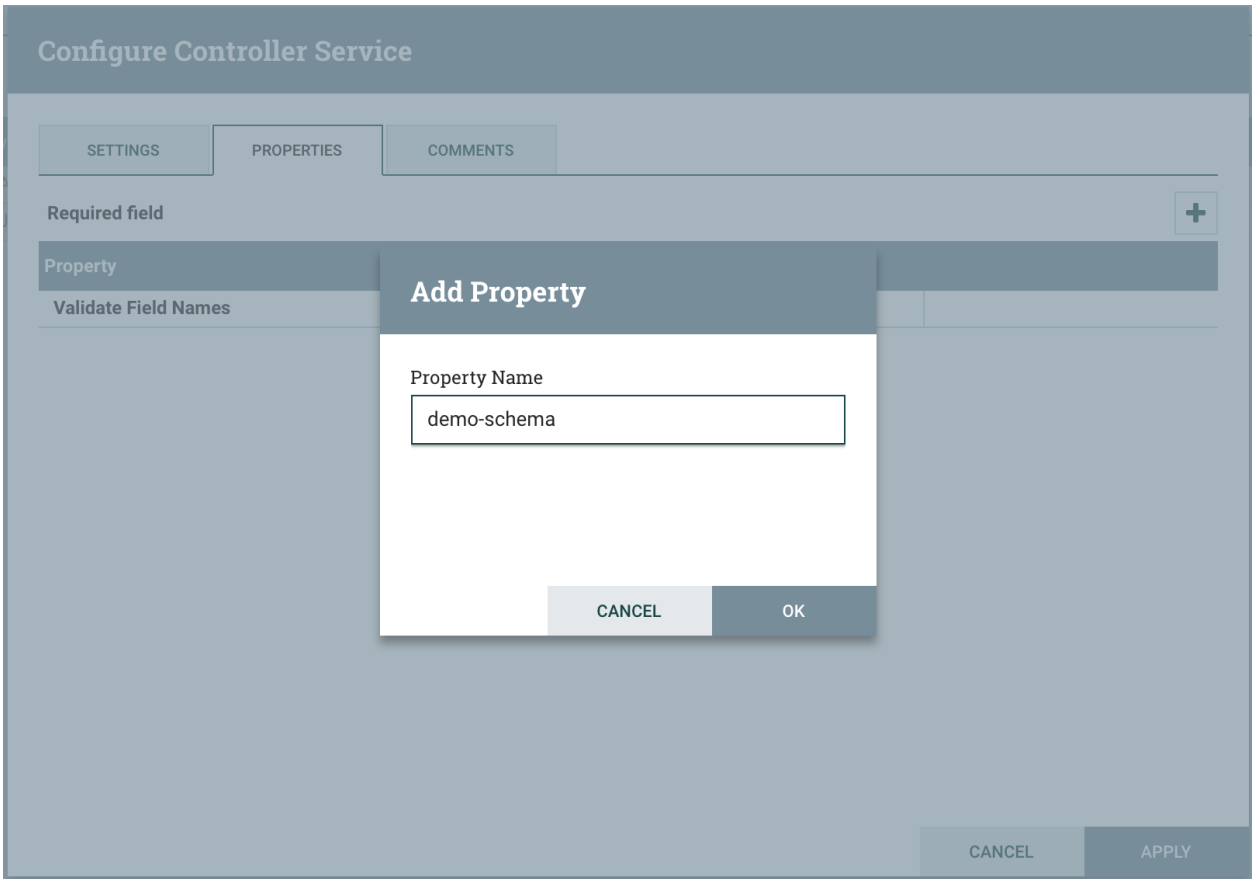
Click **YES**



Click on the **Configure** icon for the **AvroSchemaRegistryIngestion**, then click on the plus sign

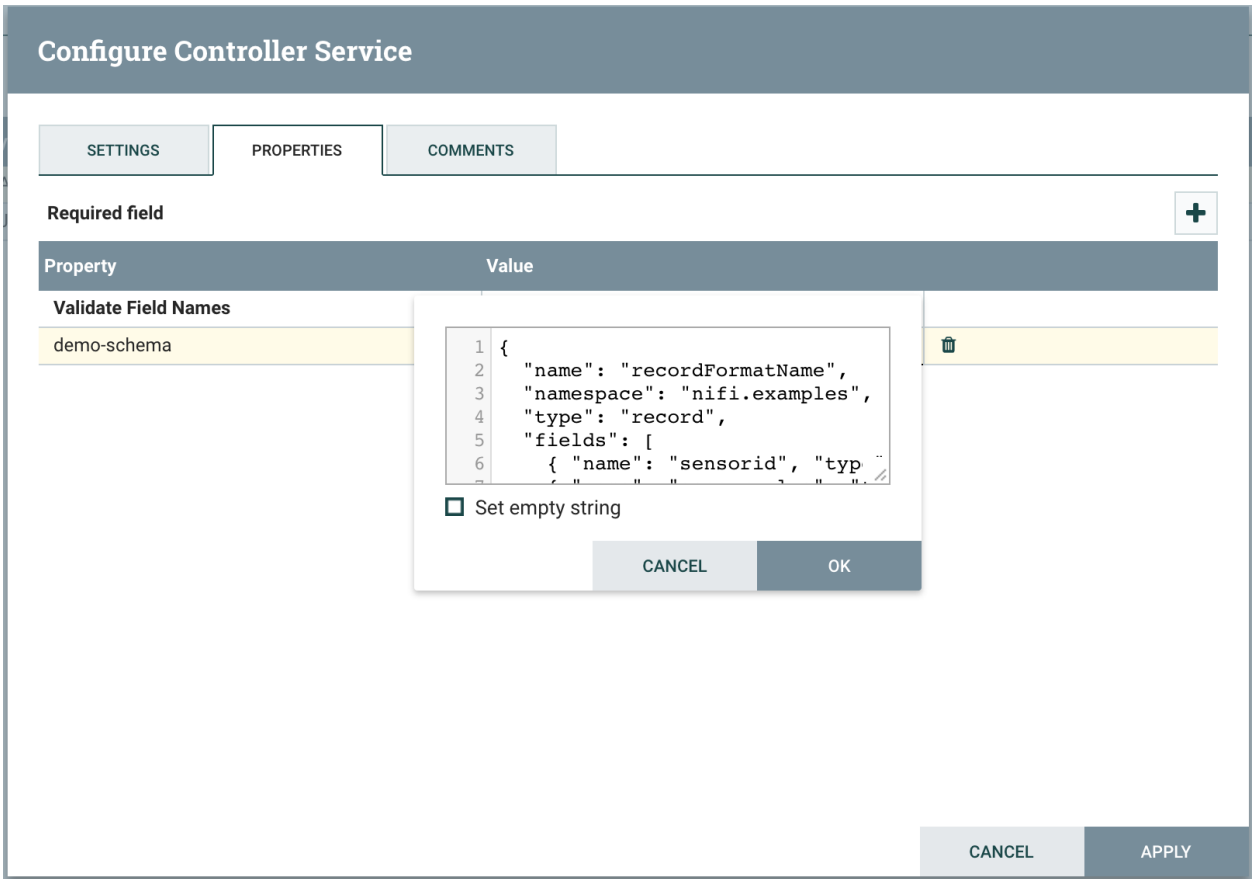


Add *demo-schema*

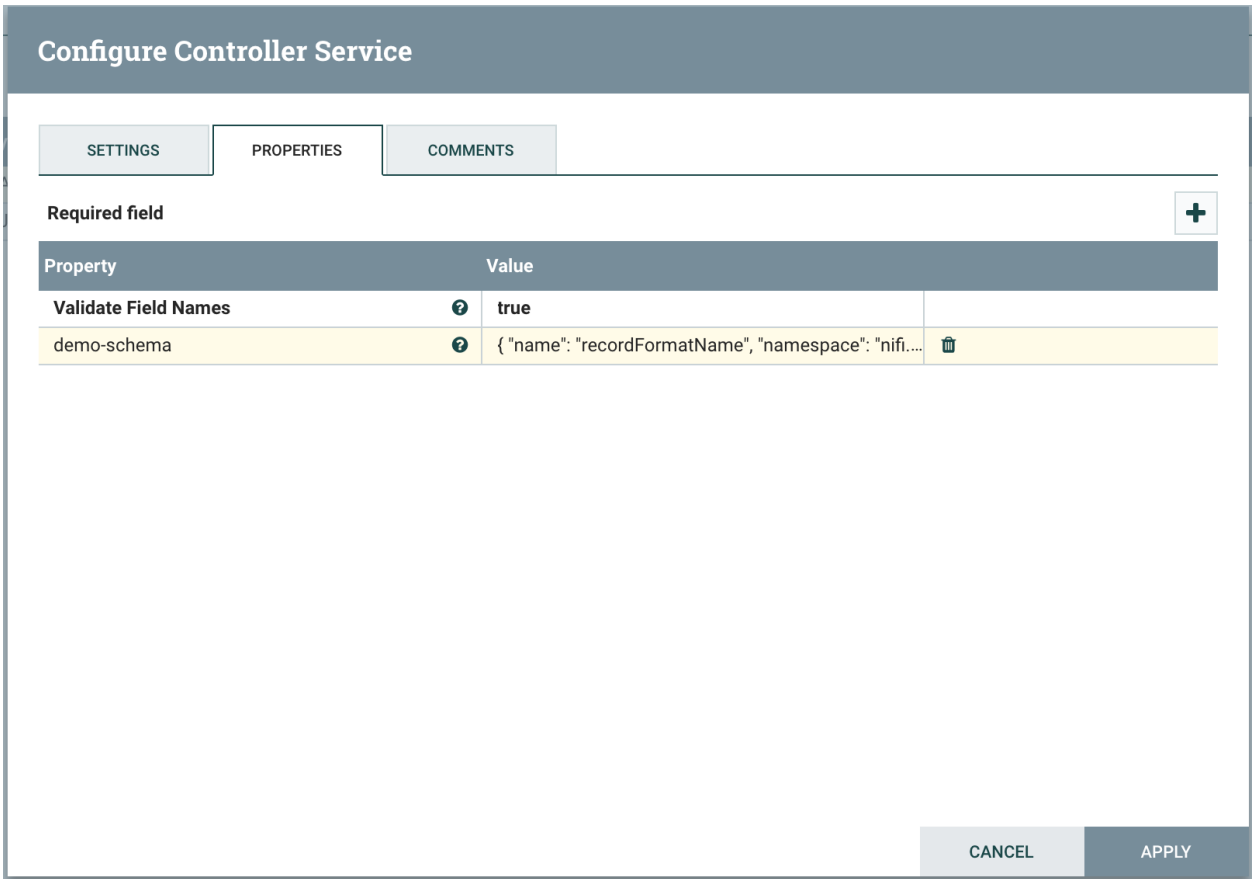


Then click **OK**.
As the Value add the following:

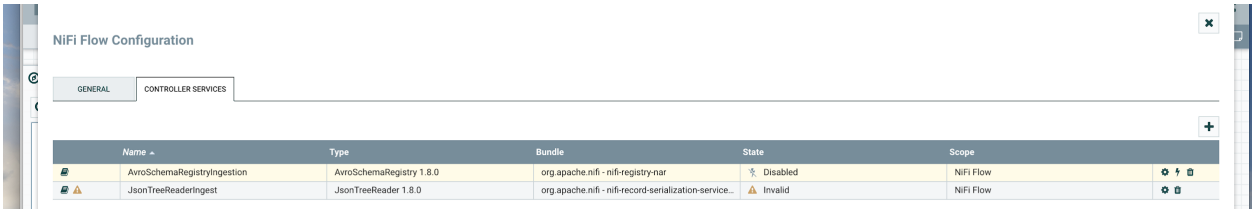
```
{
 "name": "recordFormatName",
 "namespace": "nifi.examples",
 "type": "record",
 "fields": [
   { "name": "sensorid", "type": "double" },
   { "name": "sensorvalue", "type": "double" },
   { "name": "sensortimestamp", "type": "double" }
 ]
}
```


Configure Controller Service
SETTINGS
PROPERTIES
COMMENTS
Required field
Property
Value
Validate Field Names
demo-schema
{
"name": "recordFormatName",
"namespace": "nifi.examples",
"type": "record",
"fields": [
{ "name": "sensorid", "typ
Set empty string
CANCEL
OK
CANCEL
APPLY

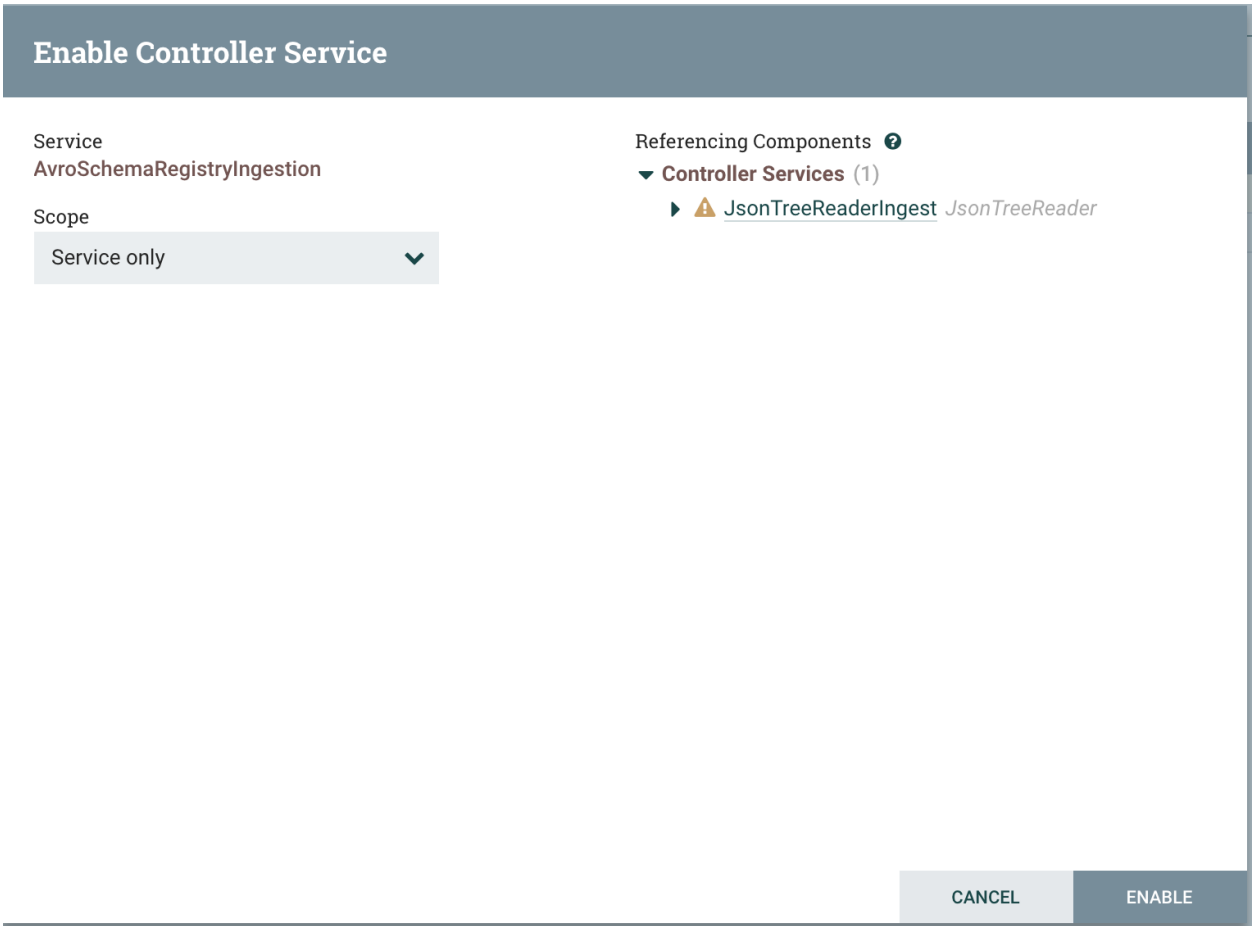
And click **OK**



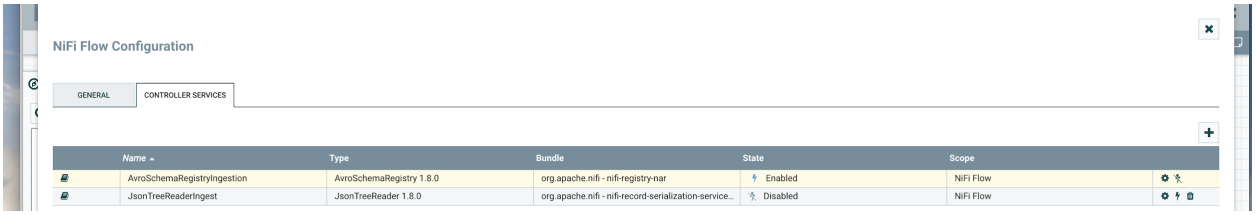
Click **APPLY**



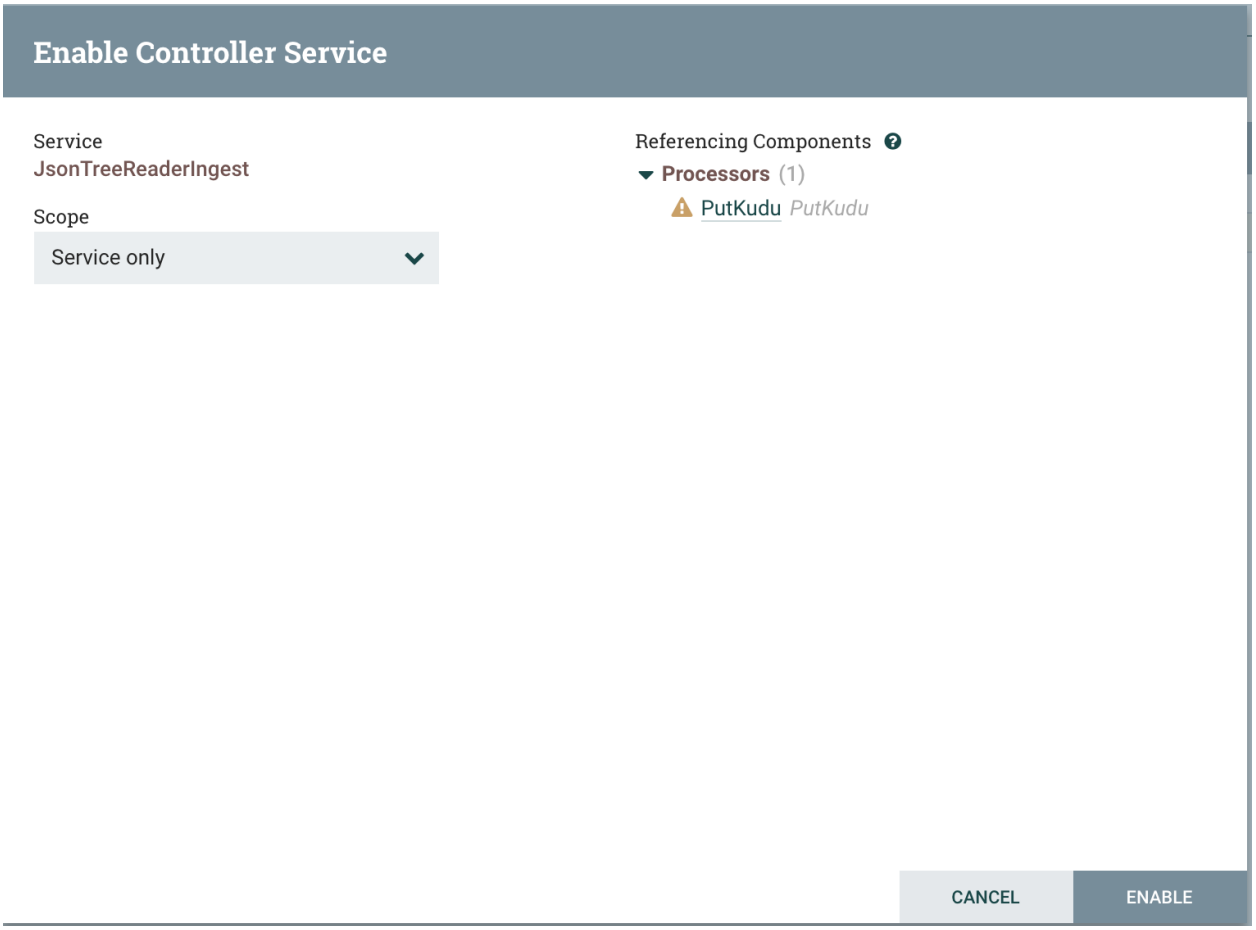
On the right, click the **Enable** icon for the **AvroSchemaRegistryIngestion**



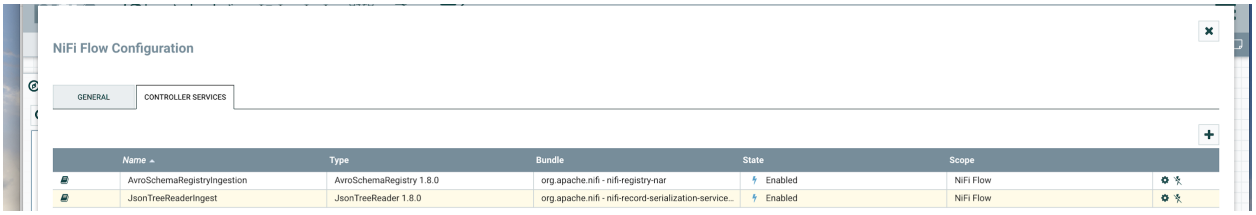
Click **ENABLE** and then **CLOSE**, and the warning icon should disappear



Then enable also the **JsonTreeReaderIngest**

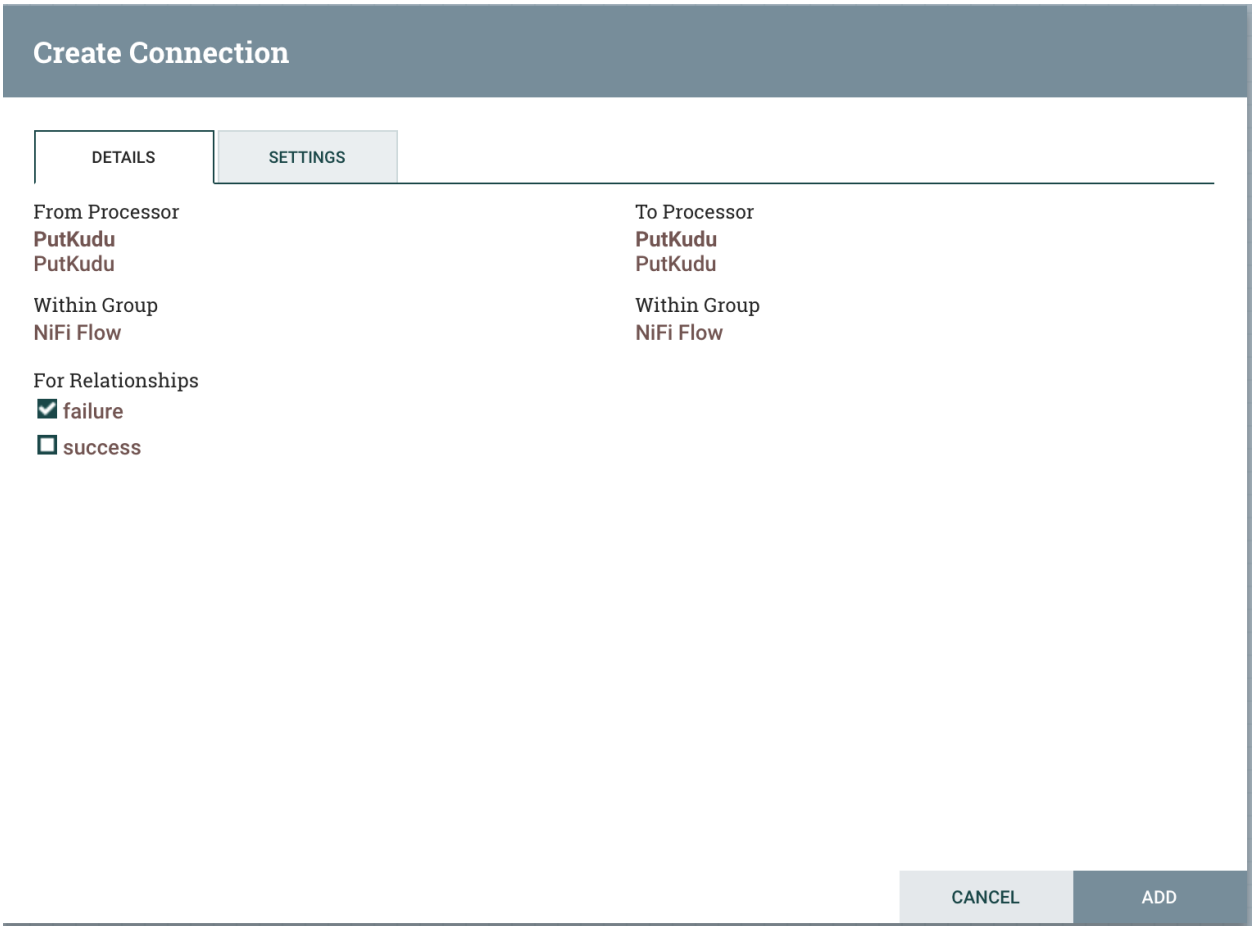


Click **ENABLE** and **CLOSE**

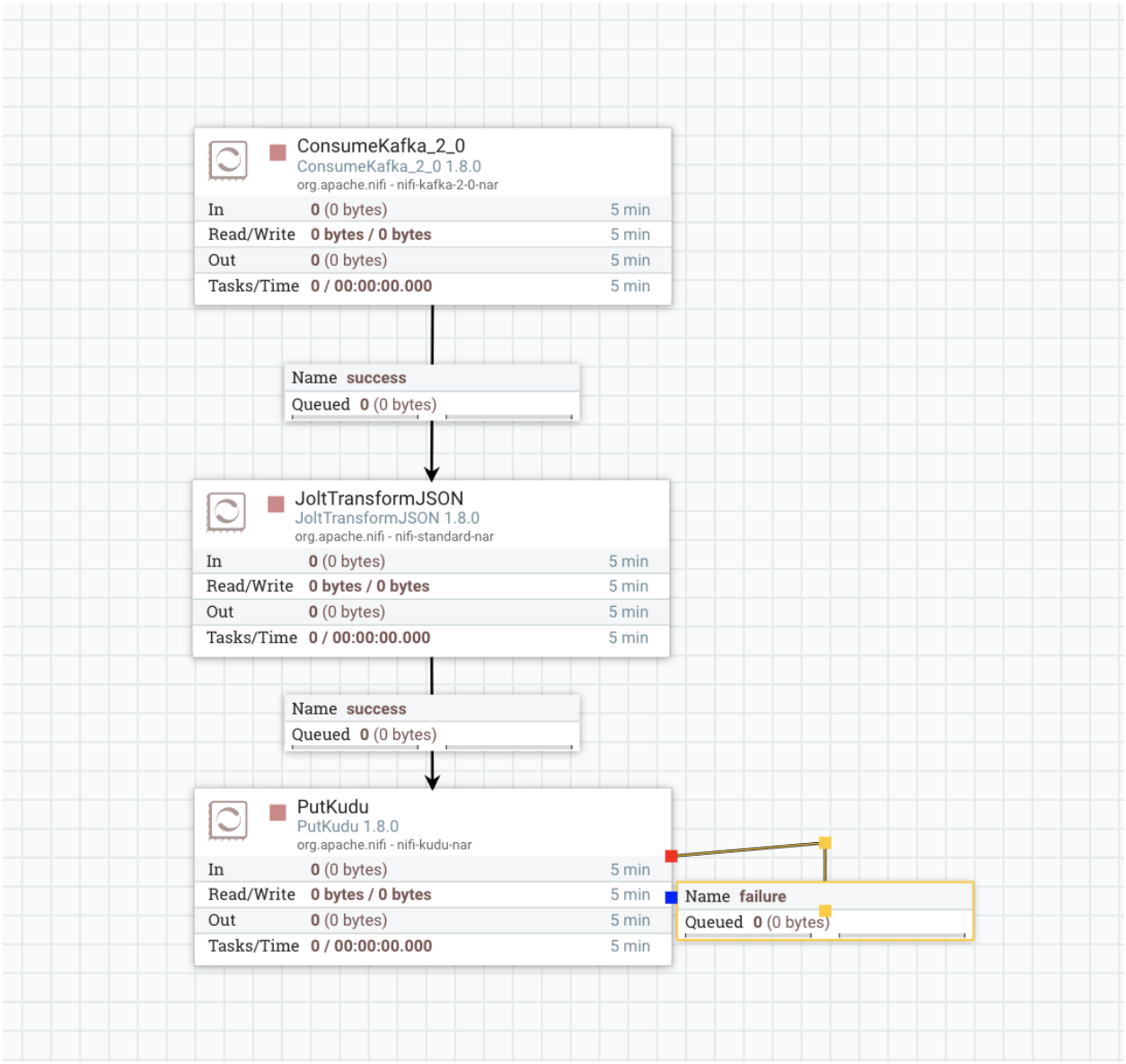


Close the **NiFi Flow Configuration** with the top right **X** icon

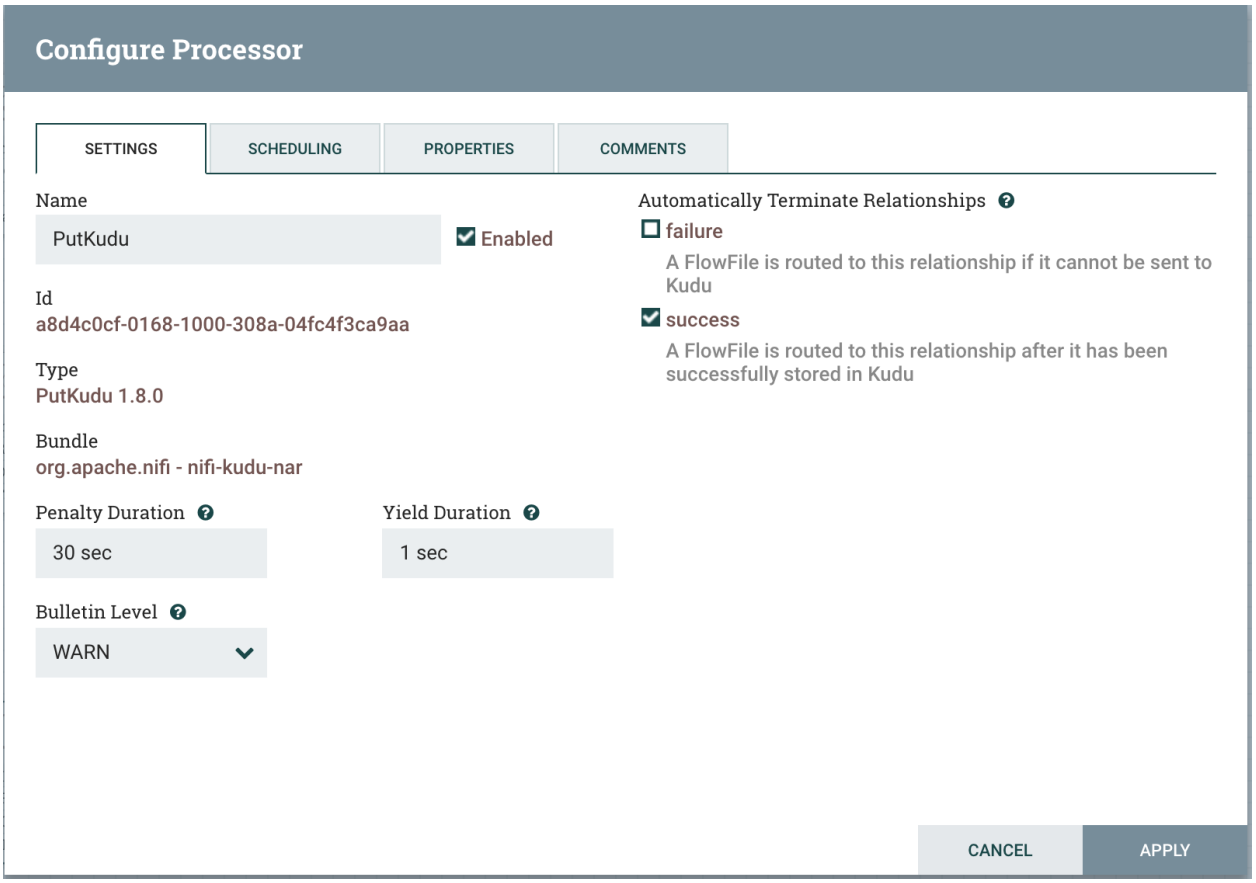
Now drag a connection from the PutKudu to itself



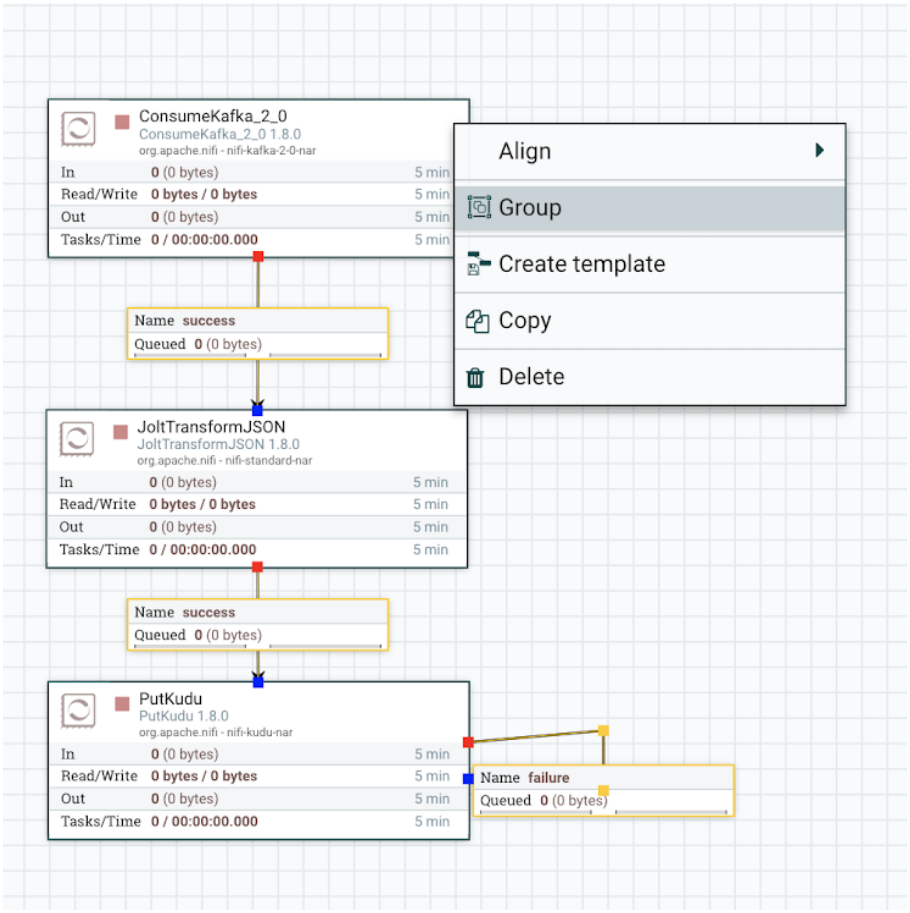
Then click **ADD**



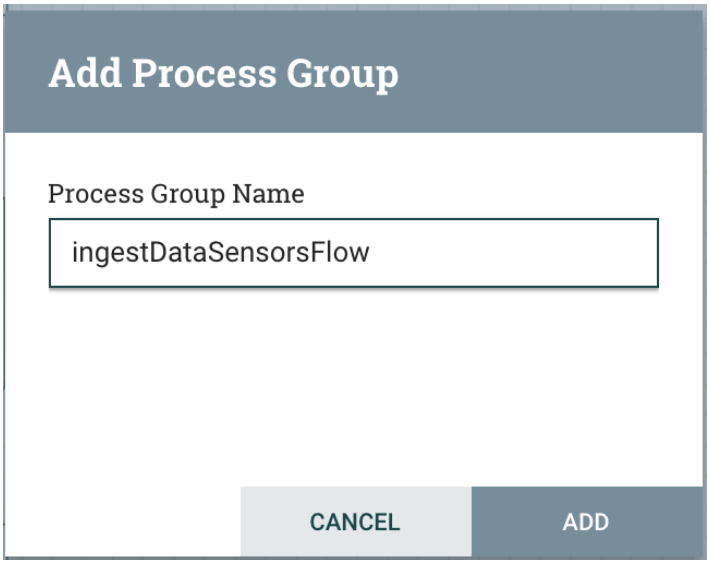
Remember also to check in the PutKudu step the **Automatically Terminate Relationships** in case of success:



Now the flow is ready to be run, but before to run it, let's create a **Group**.
Do you remember how to do it? SHIFT hold…



Give it a name, in my case ingestDataSensorsFlow:

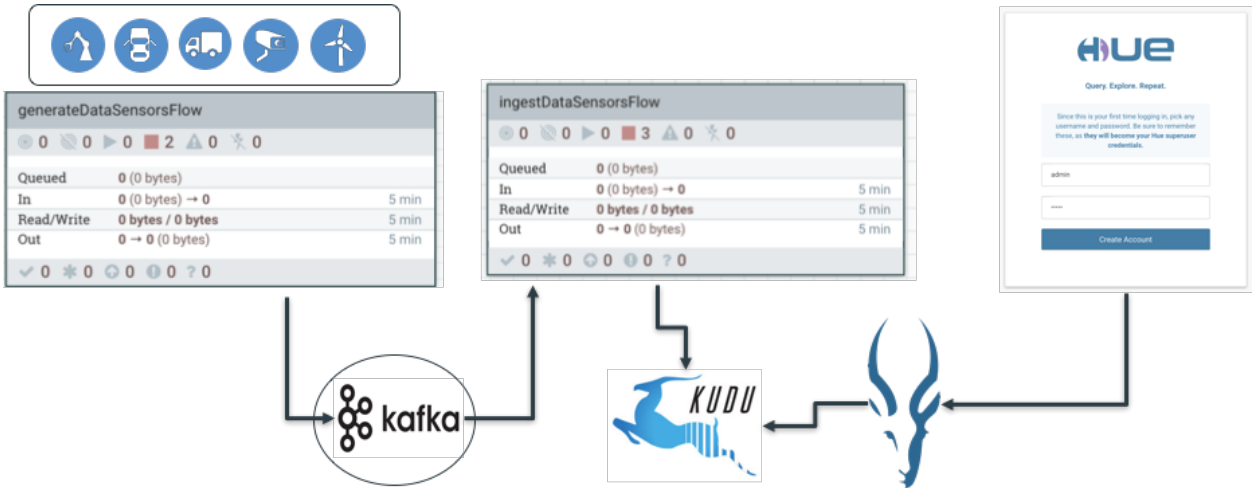


And here our group

Now we are ready to test the end2end flow.

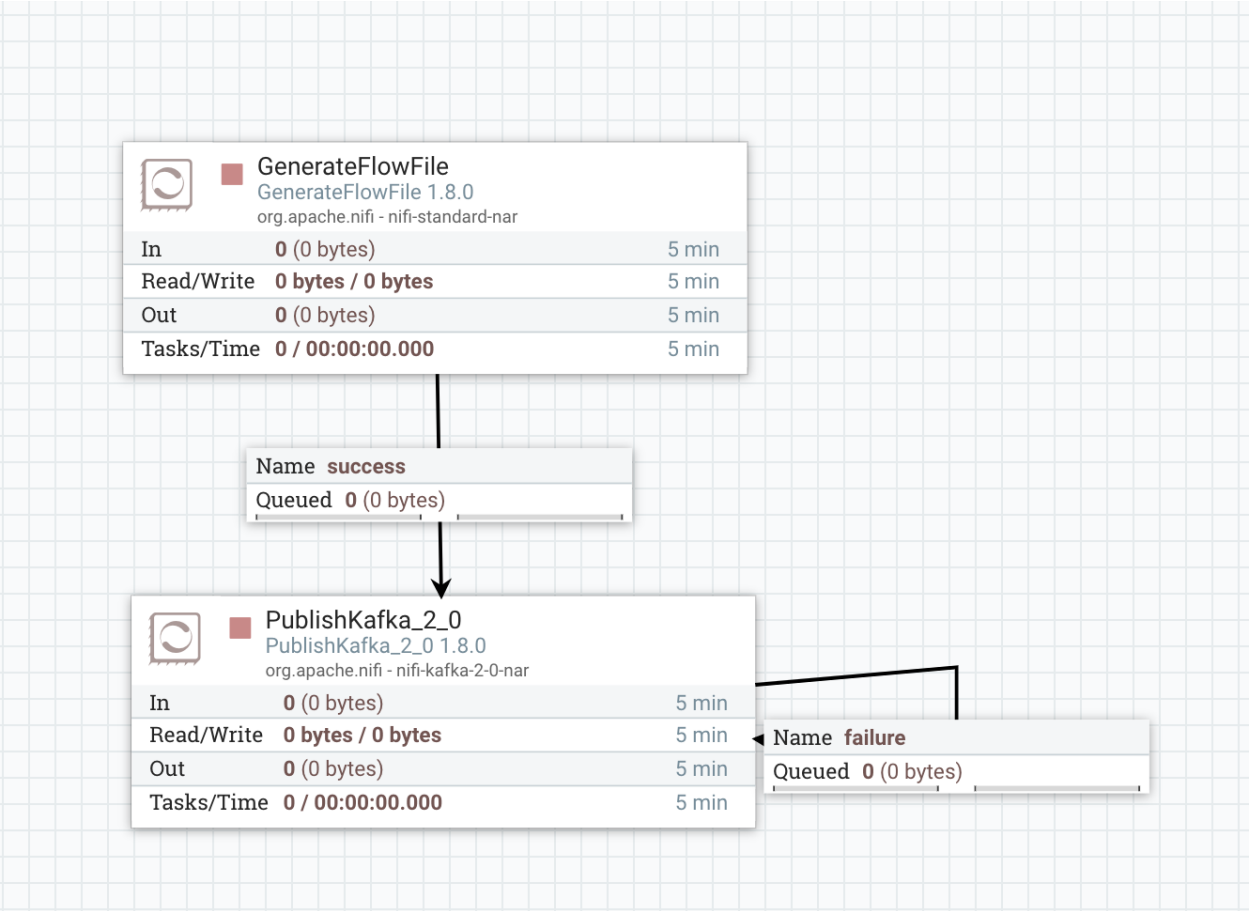
Lab 6: Run and verify the real-time ingest pipelines

At this point we have built two pipelines in NiFi. The first one feeds into a Kafka topic; the second one consumes from that topic and writes to Kudu:

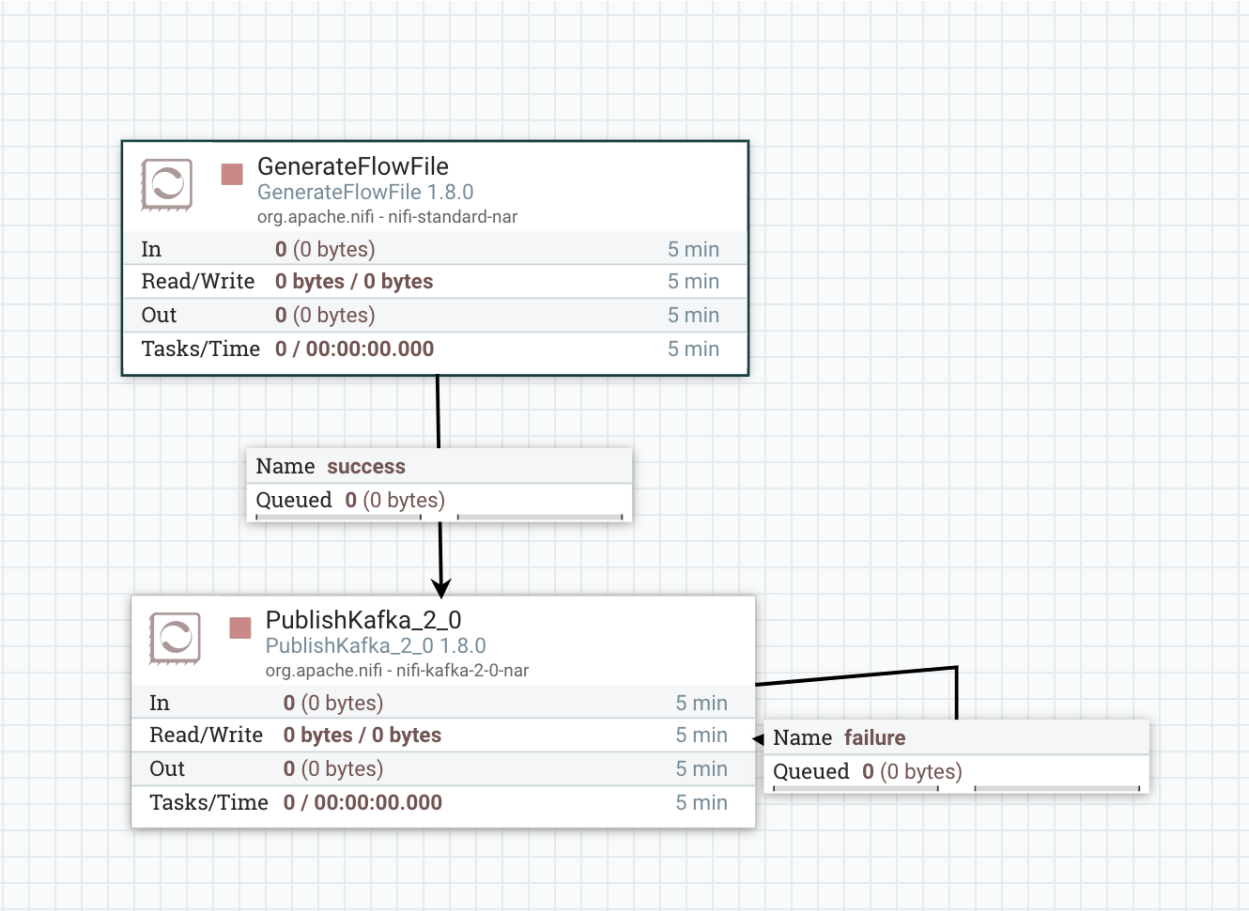


NiFi allows you a fine control on which components in your flows to run, this is very useful when debugging and testing the flows.

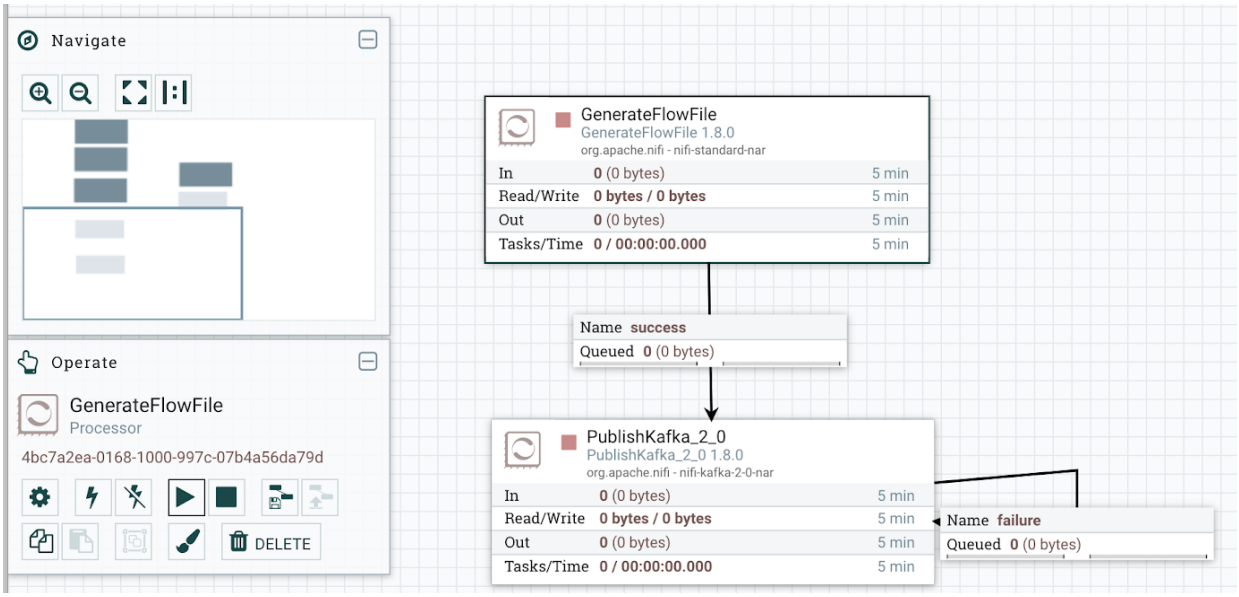
Let's start with the first flow, the one that generate the random sensors readings:



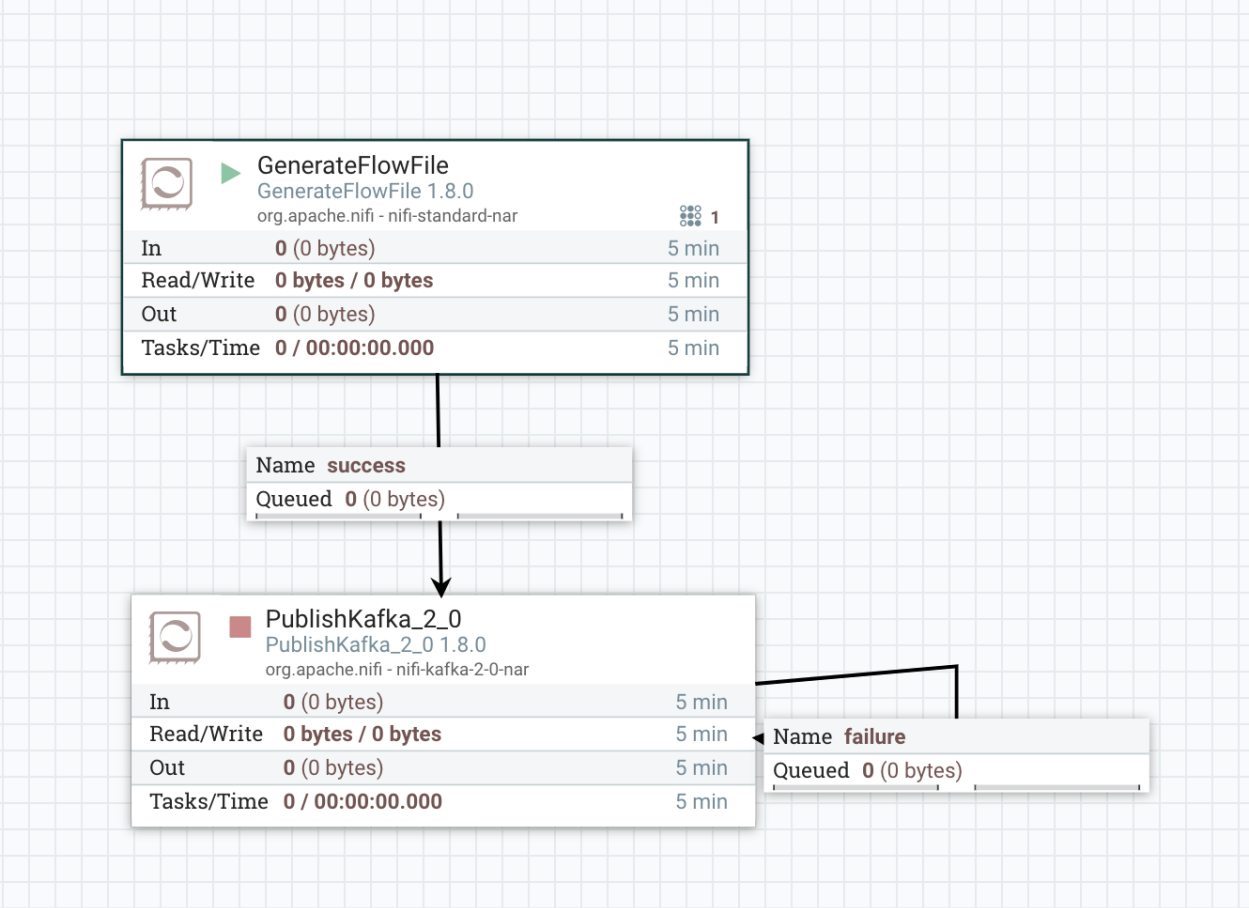
Select the first step:



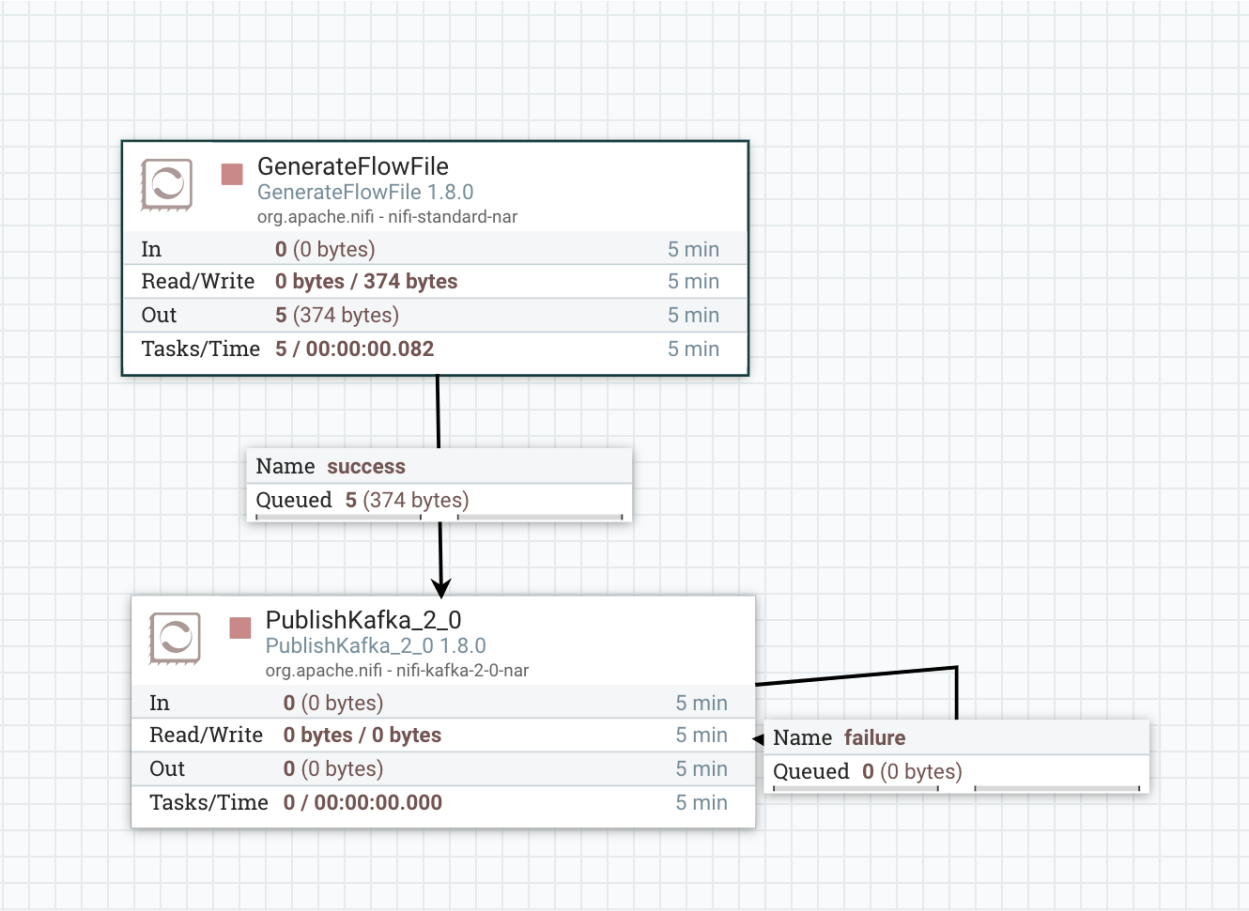
Now, in the **Operate** box on the left click the **Start** icon



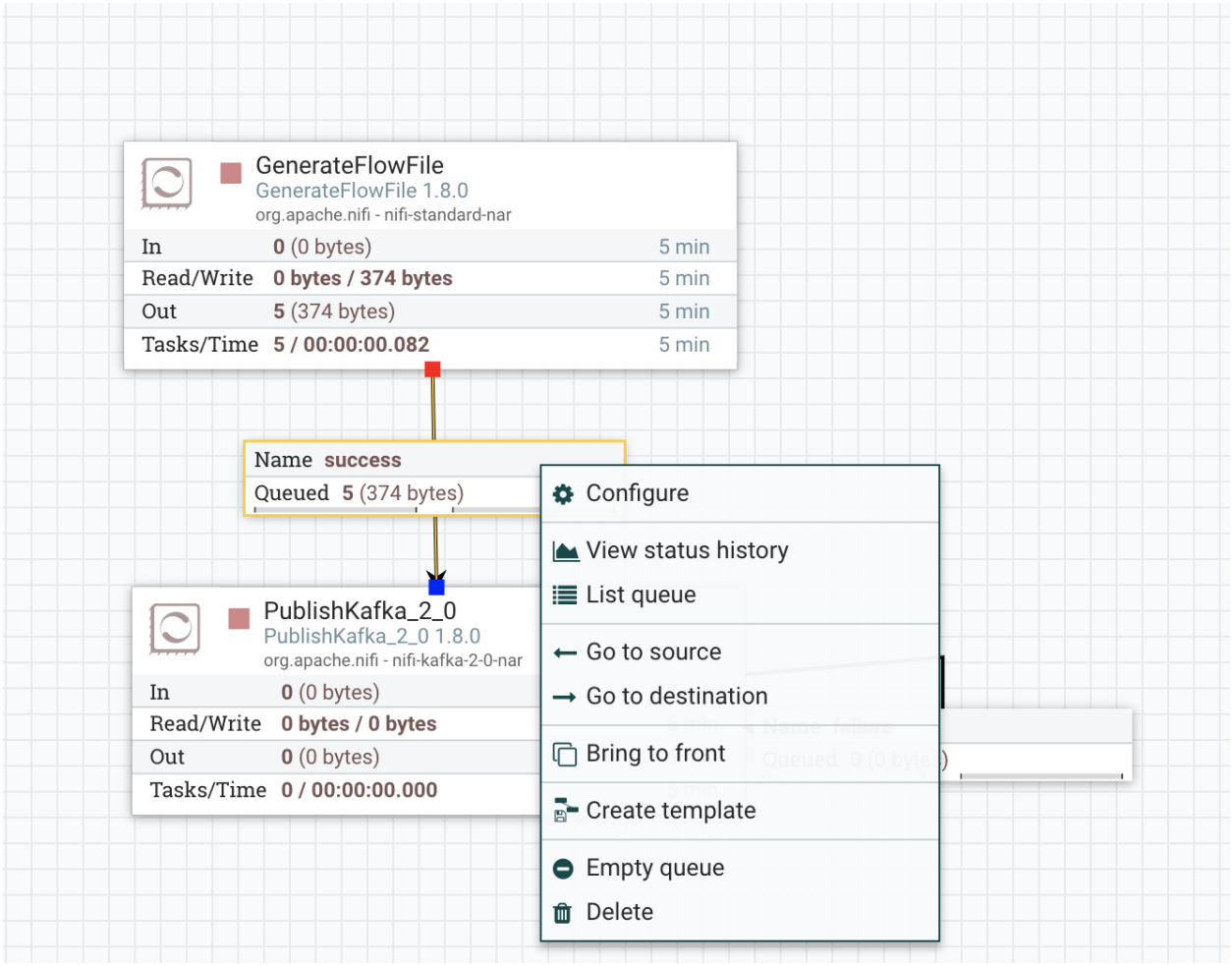
Now you should see the **Run** icon on the top left corner of the **GenerateFlowFile**:



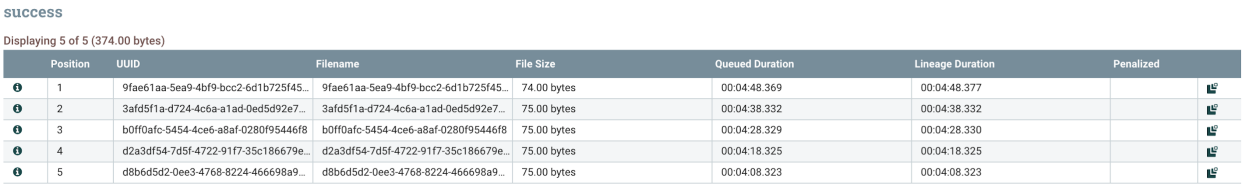
Let it run until you see some messages in the **Queue**, in my case I've 5 messages accumulated in the queue. You can now stop the GenerateFlowFile processor.



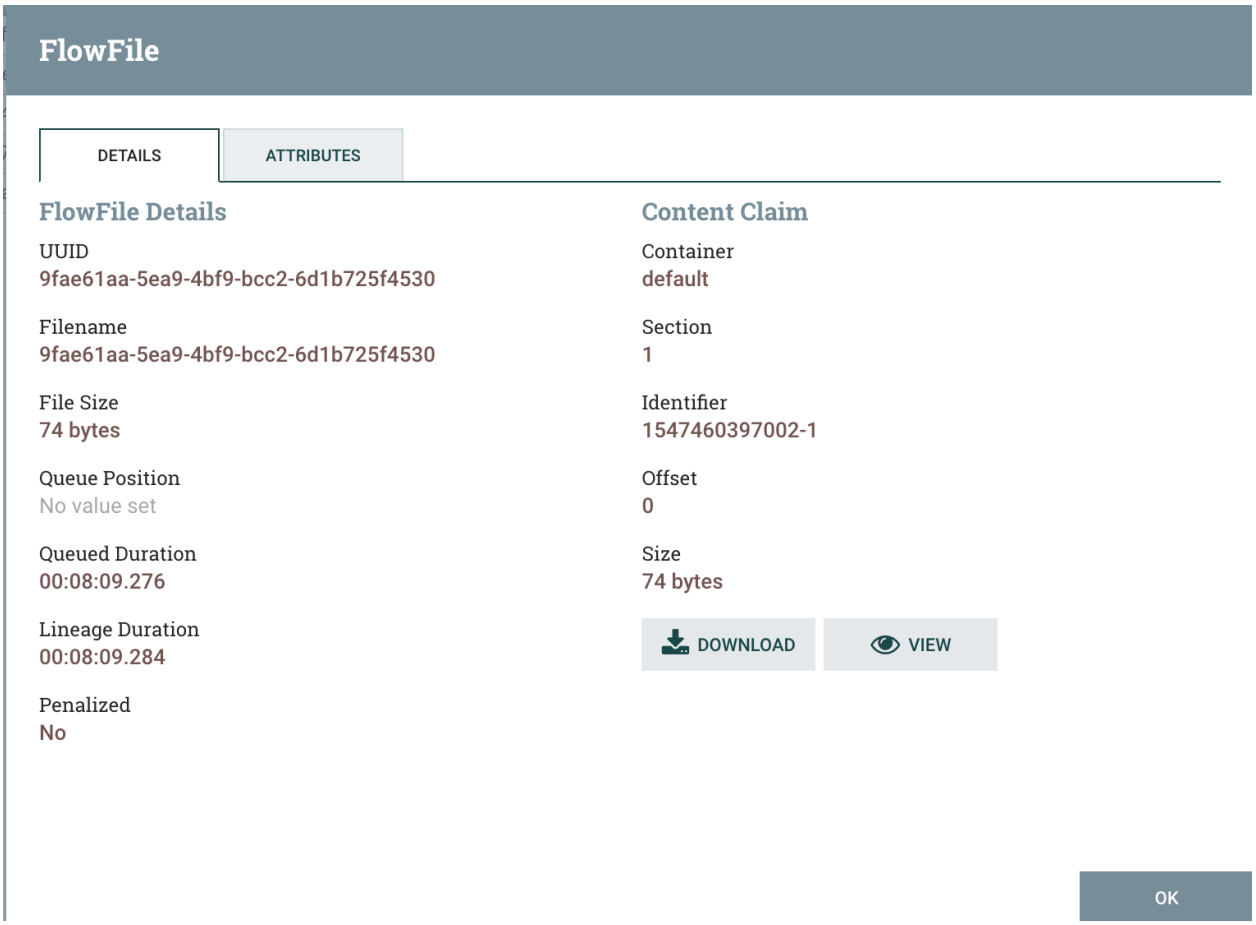
Select the **Queue** and right click, then select **List queue** option:



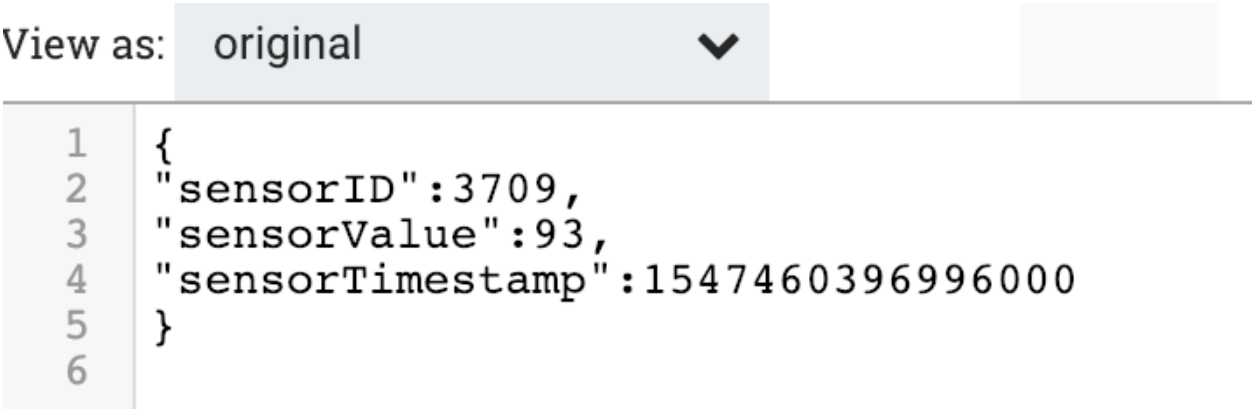
You should see the list of all the message awaiting in the queue:



Choose one message and click on the **Information** icon, in order to view the details

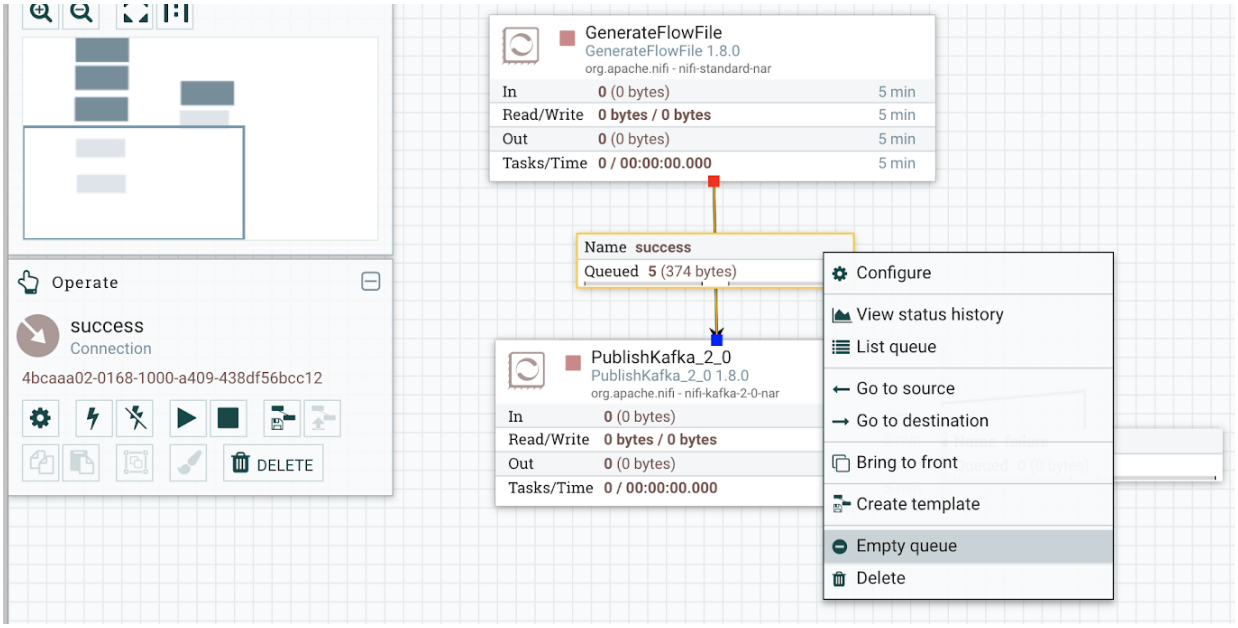


Then click on the **VIEW** icon

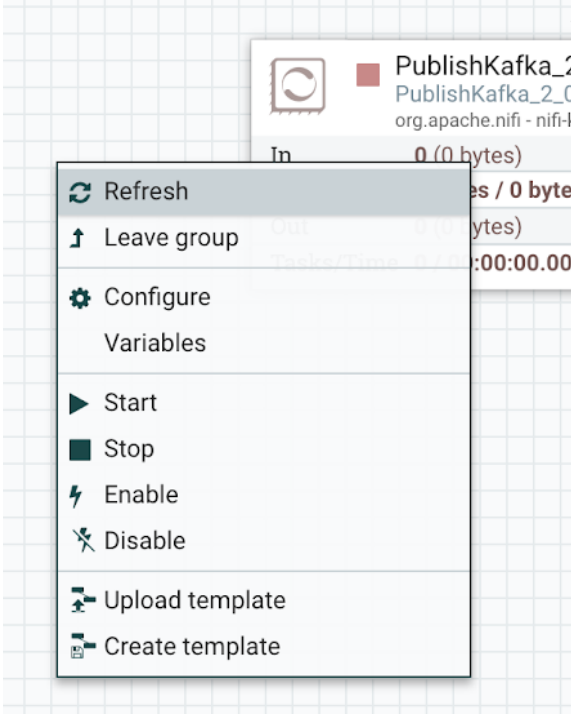


This is exactly the message format that we wanted to generate. Double check other messages in order to see if they are conform to the same format.

We could empty the queue **but do not do it**; empty the queue maybe useful when debugging your flow you have realized that the messages are not corrected for the downstream steps:



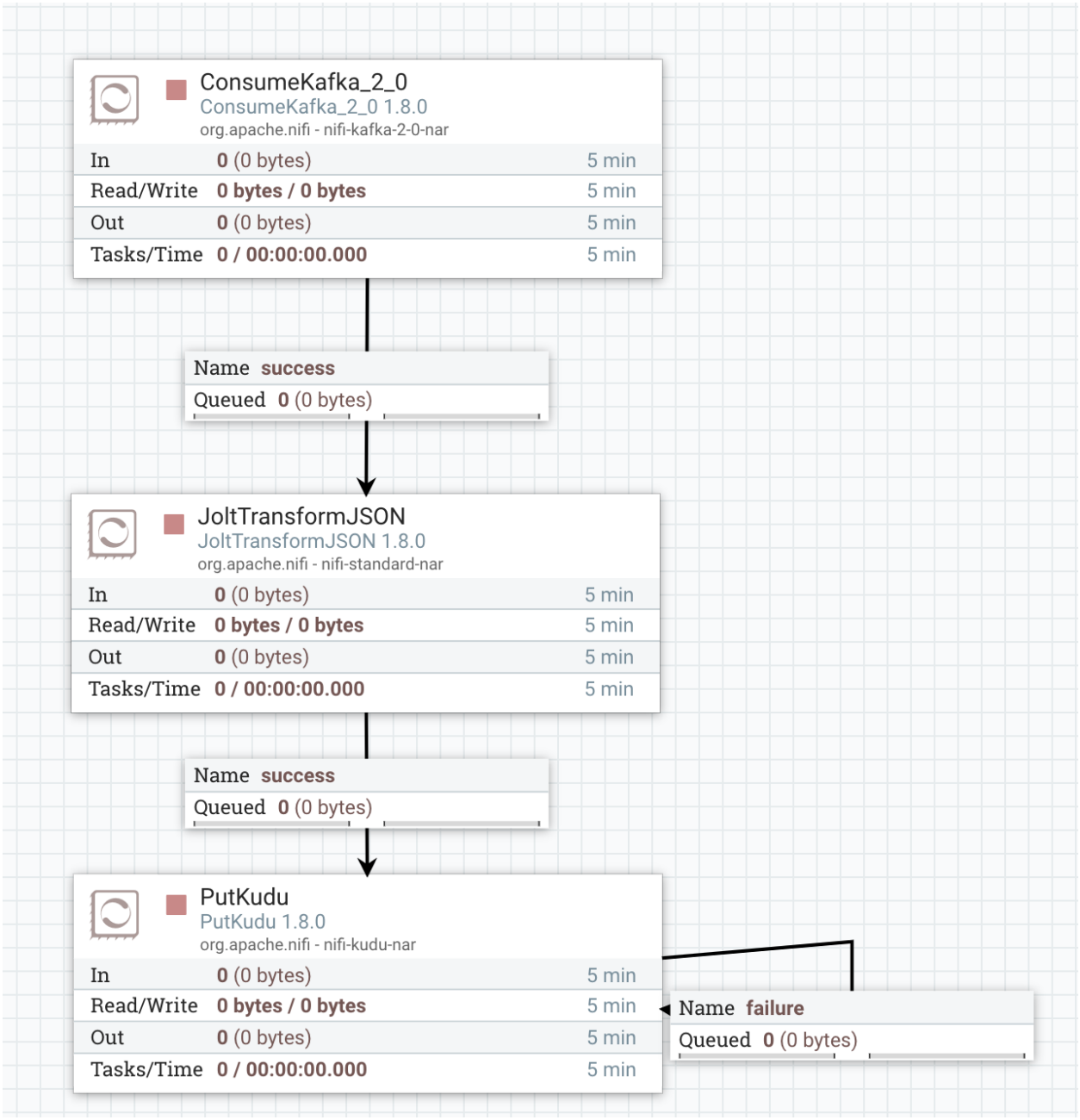
We want to see if the kafka components are working fine. So in order to to that go to the second flow, the one that ingest the data. In order to do it you can click in an empty point in the canvas:



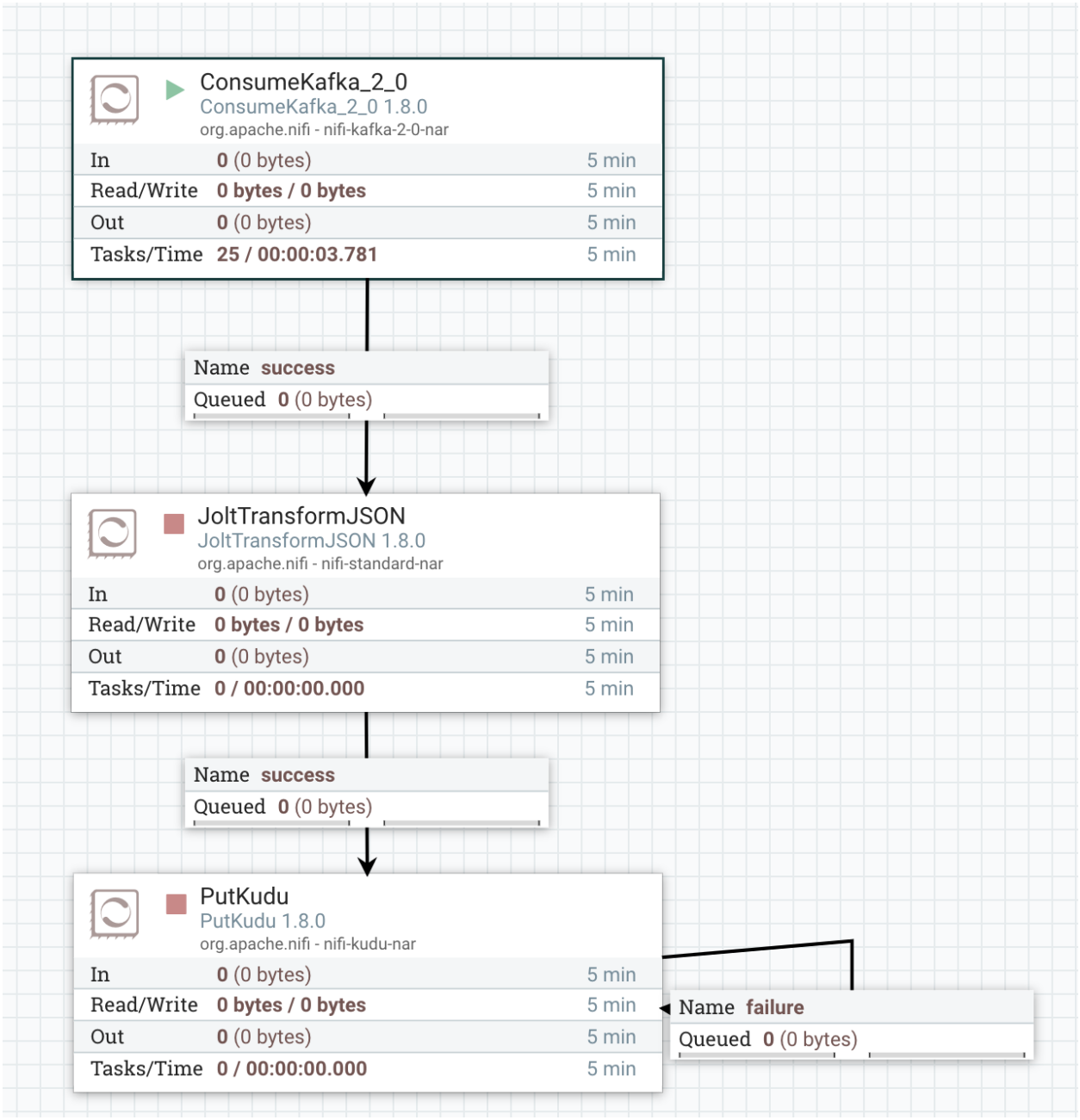
And click **Leave group**

Alternatively, you can also click the NiFi link at the bottom of the web UI:

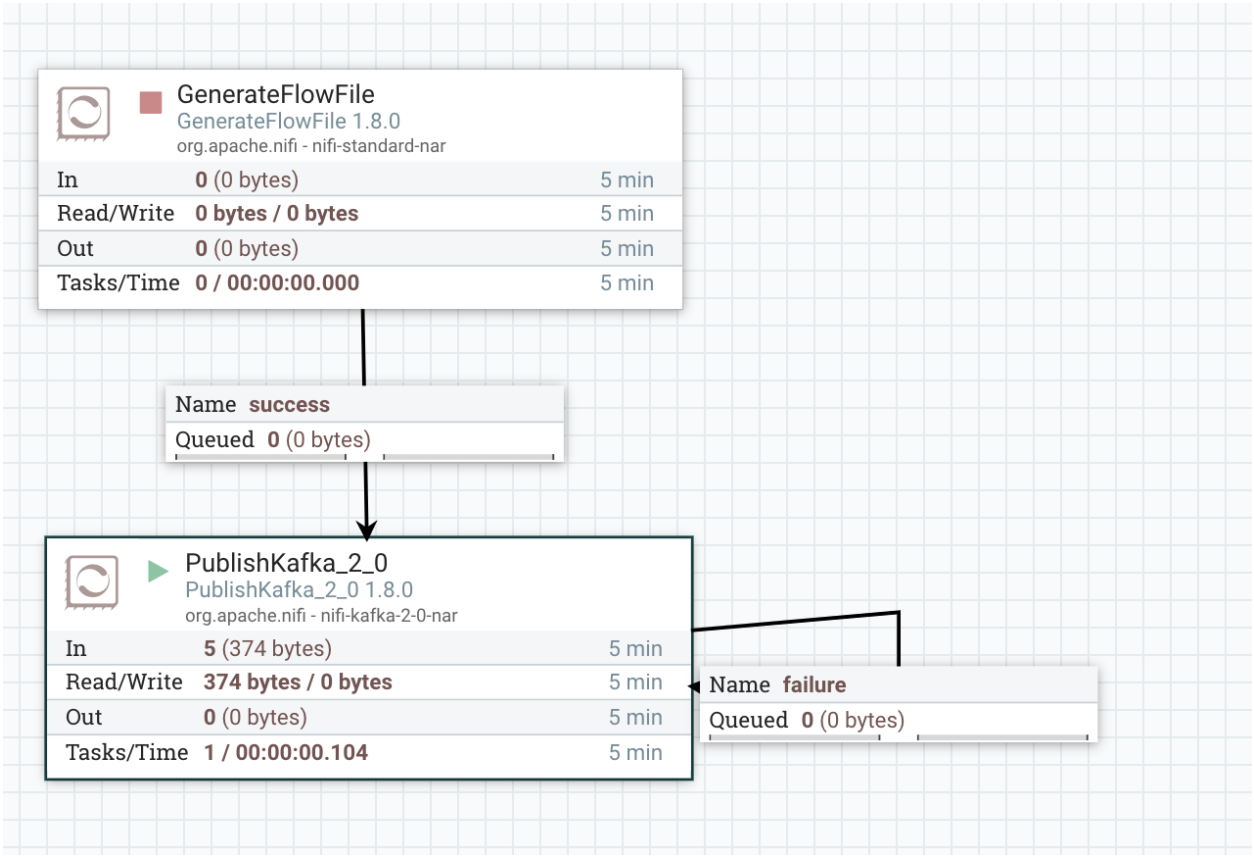
And the go to the *ingestDataSensorsFlow*:



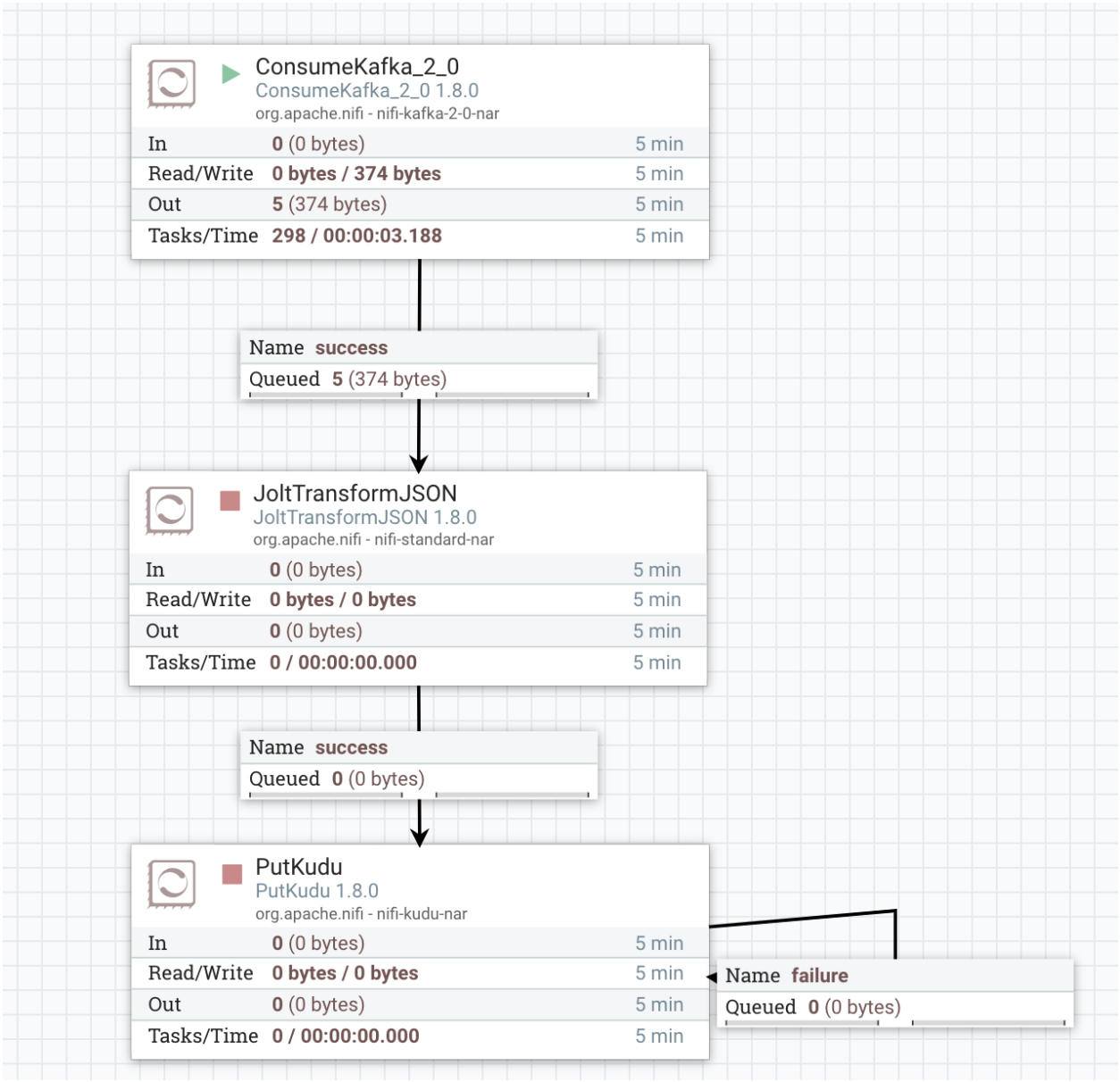
And run the **ConsumeKafka** processor:



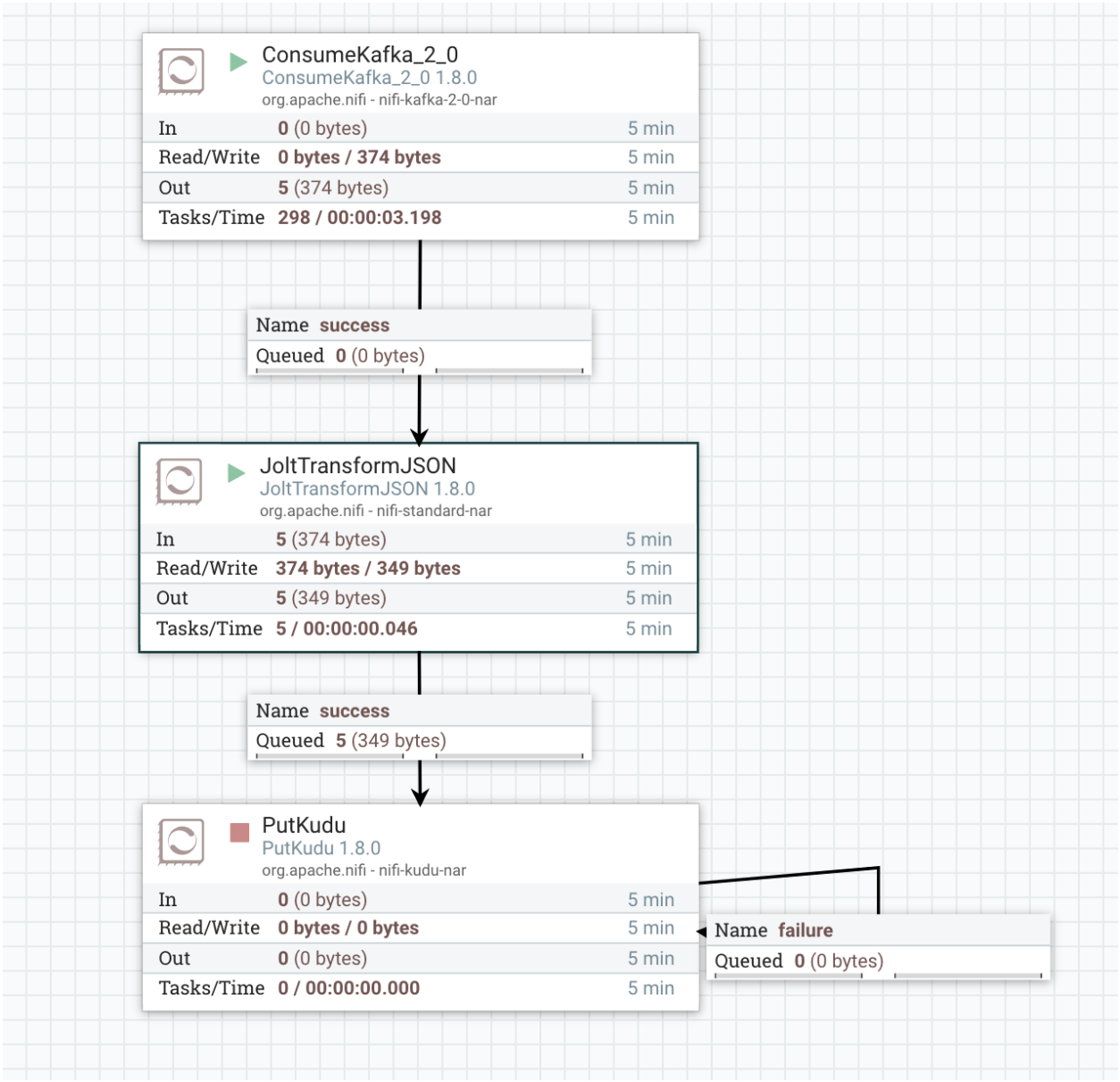
It does not consume messages because the **PublishKafka** component on the first flow is still in **Stop** state. Go back to the first flow and run the **PublishKafka**



As you can see the Queue now is empty. Go back to the ingest flow:



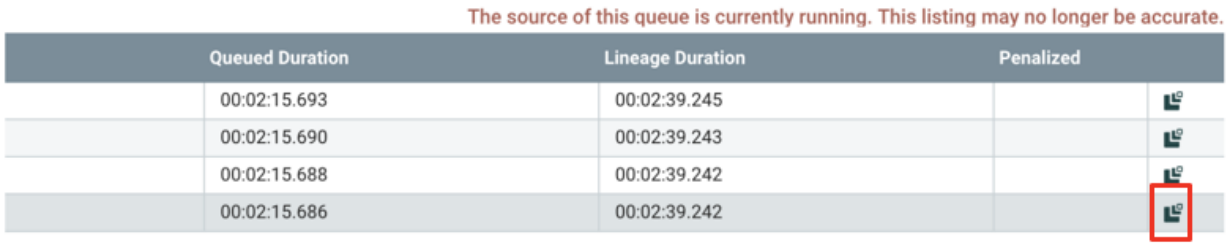
As you can notice, now we have 5 messages in this queue. Check the content of the messages. Run now the **JoltTransformJSON**:



Now the 5 messages are in the queue in front of the **PutKudu** steps. Check the content of some message to see the transformation worked properly:

The fields are all in lowercase, as we were expected.

You can also check the provenance feature of NiFi.
Go back to the queue list, and click on the Provenance icon below:



The source of this queue is currently running. This listing may no longer be accurate.

Queued Duration	Lineage Duration	Penalized	
00:02:15.693	00:02:39.245		
00:02:15.690	00:02:39.243		
00:02:15.688	00:02:39.242		
00:02:15.686	00:02:39.242		

NiFi Data Provenance

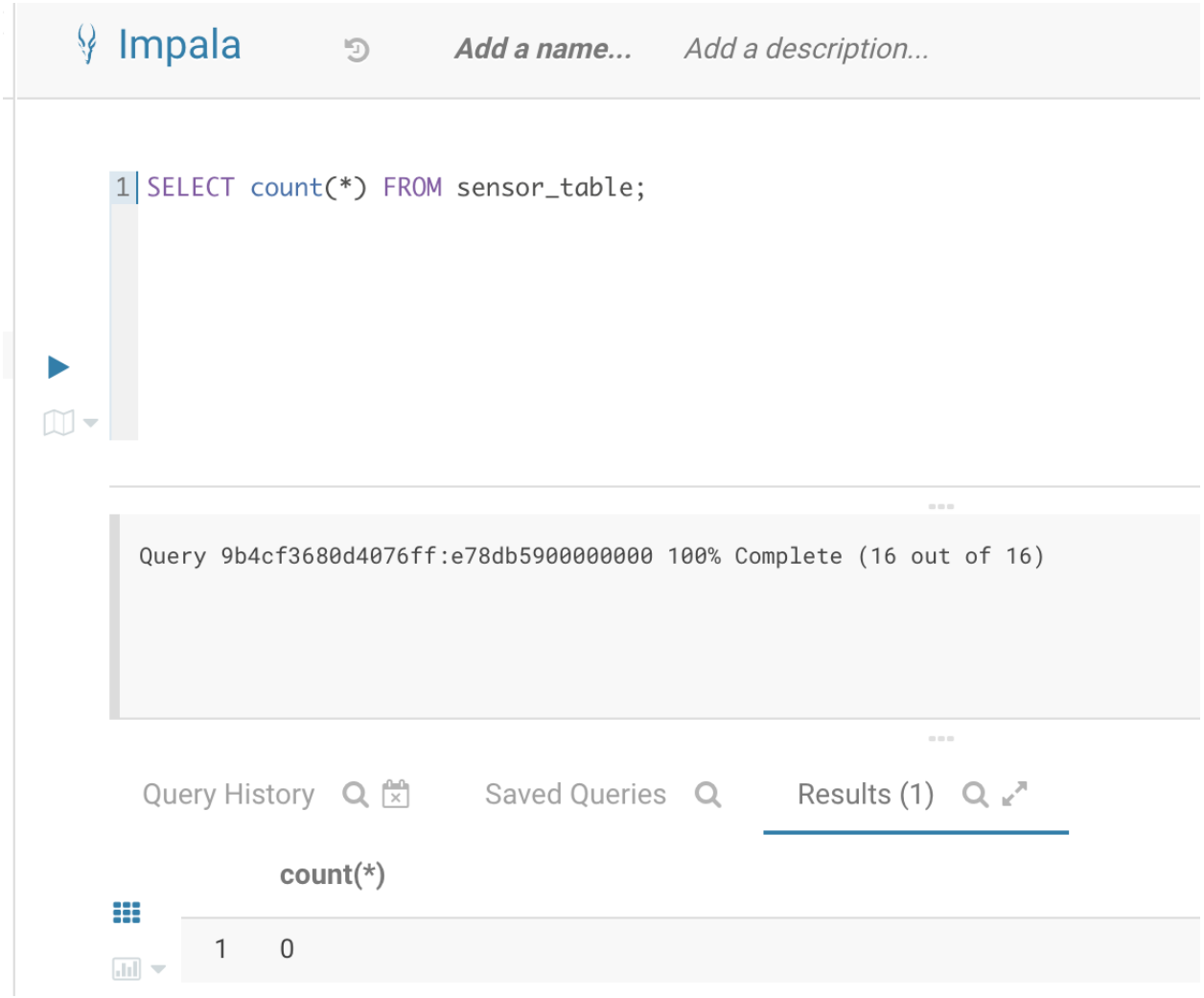
Displaying 3 of 3
Oldest event available: 01/31/2019 13:48:51 UTC

Filter | by component name

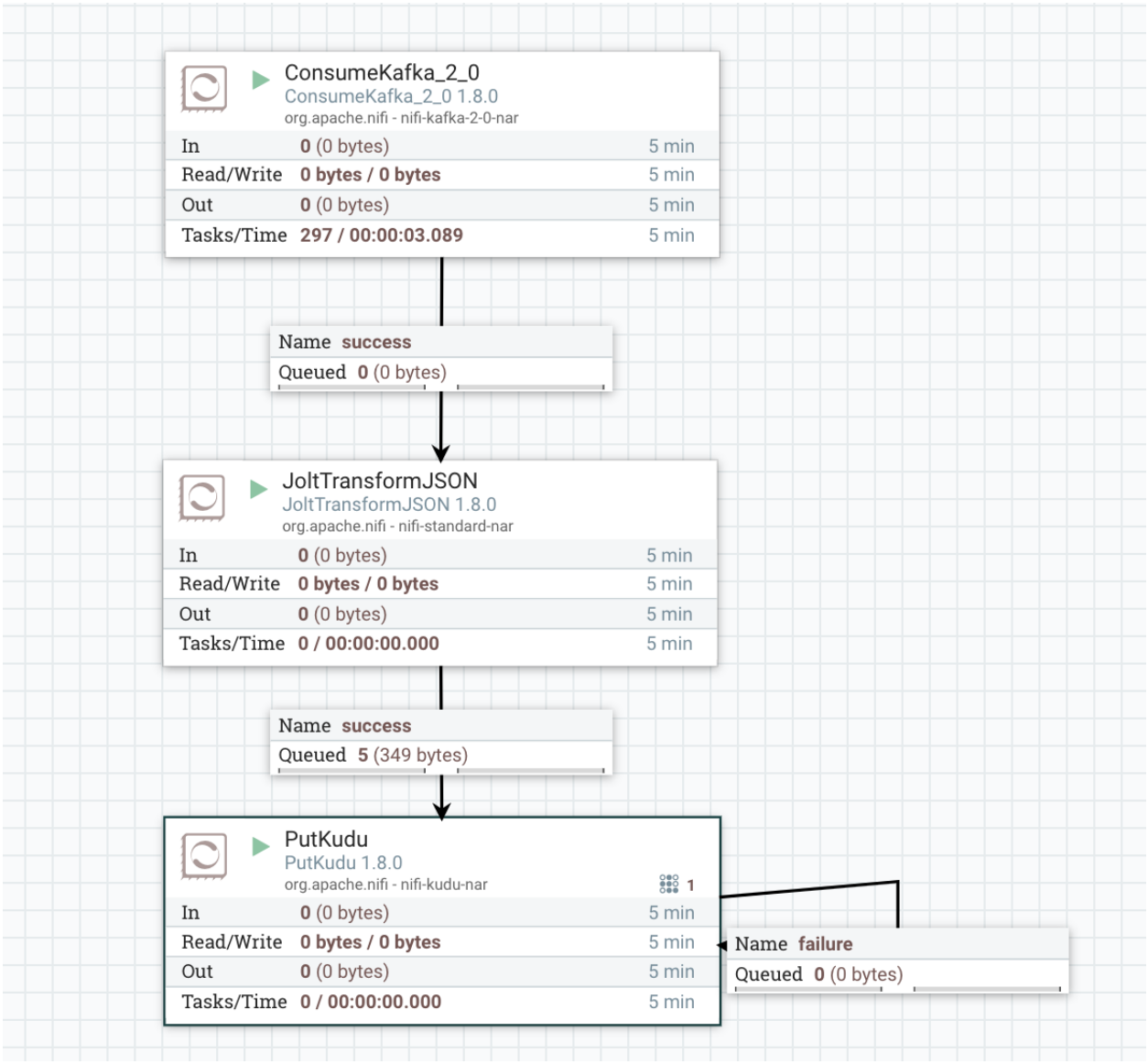
	Date/Time	Type	FlowFile Uuid	Size	*Component Name*	Component Type
ⓘ	01/31/2019 14:02:43.753 UTC	DOWNLOAD	e94624f8-715a-44ed-90...	70 bytes	No value set	NiFi Flow
ⓘ	01/31/2019 14:01:29.464 UTC	RECEIVE	e94624f8-715a-44ed-90...	70 bytes	ConsumeKafka_2_0	ConsumeKafka_2_0
ⓘ	01/31/2019 14:01:53.020 UTC	CONTENT_MODIFIED	e94624f8-715a-44ed-90...	70 bytes	JoltTransformJSON	JoltTransformJSON

Switch to HUE interface, *<master IP>:8888* (in my case http://flamb2-mn0.westeurope.cloudapp.azure.com:8888), and run the following query to check that the sensor table is empty:

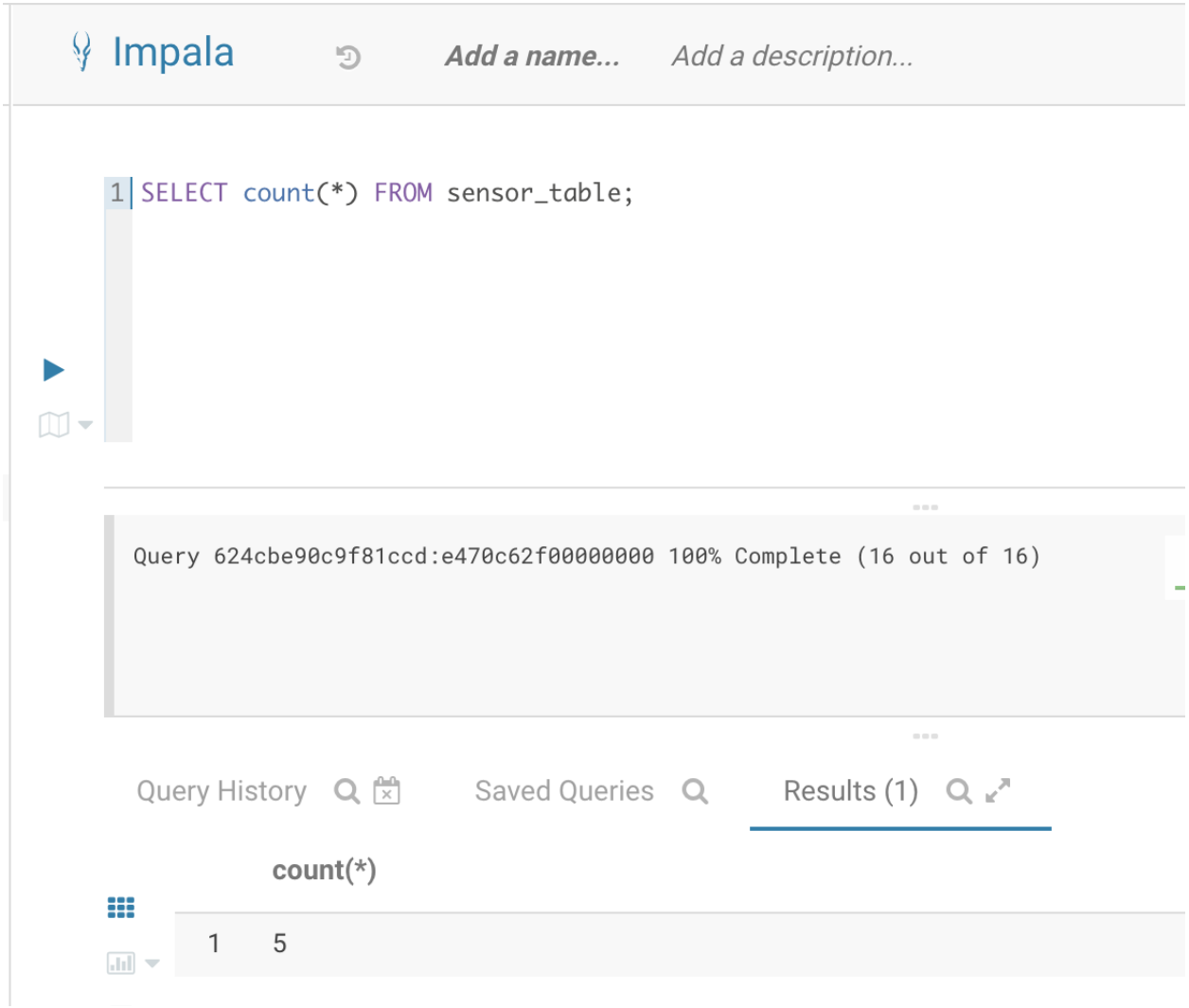
```
SELECT count(*) FROM sensor_table;
```



Now we can run the last step, the **PutKudu** processor:

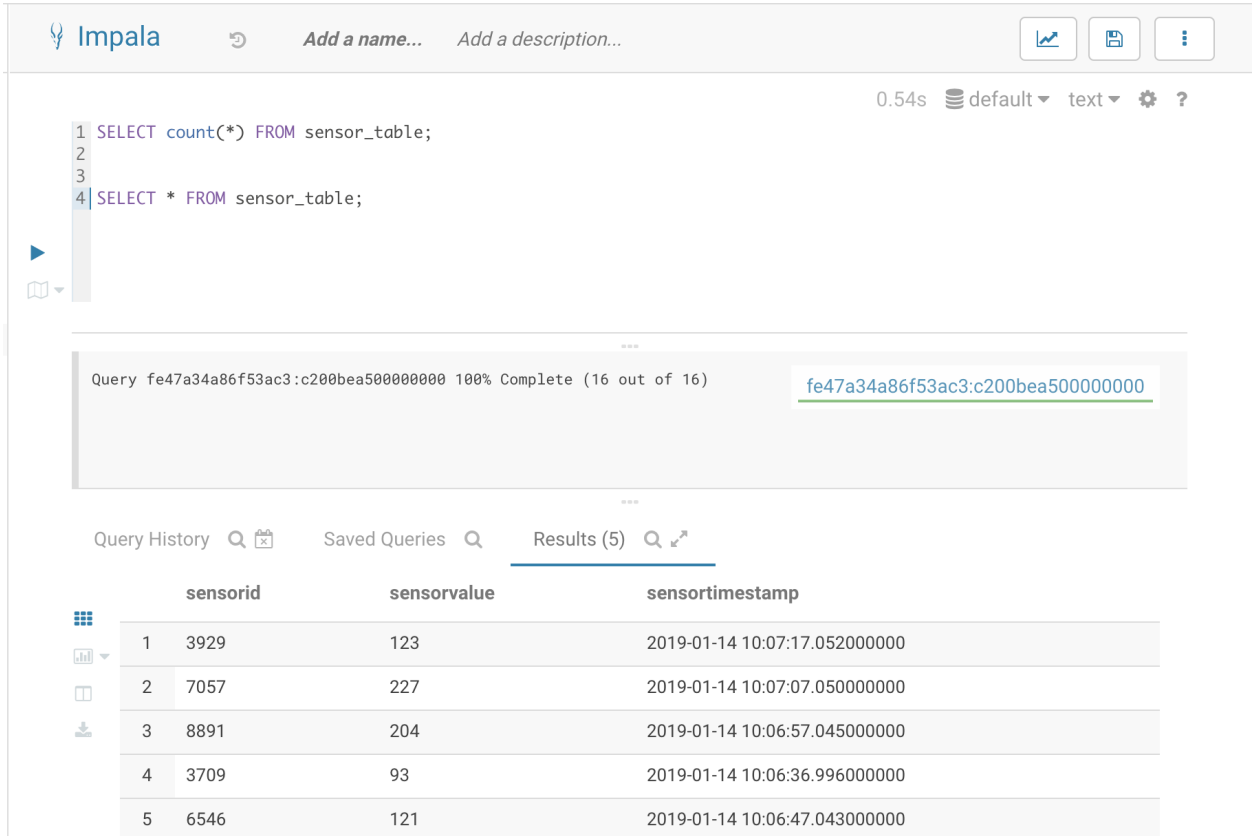


Run again the SELECT query:



You can see that we have 5 messages. Check the content of these messages with the following statement:

```
SELECT * FROM sensor_table;
```

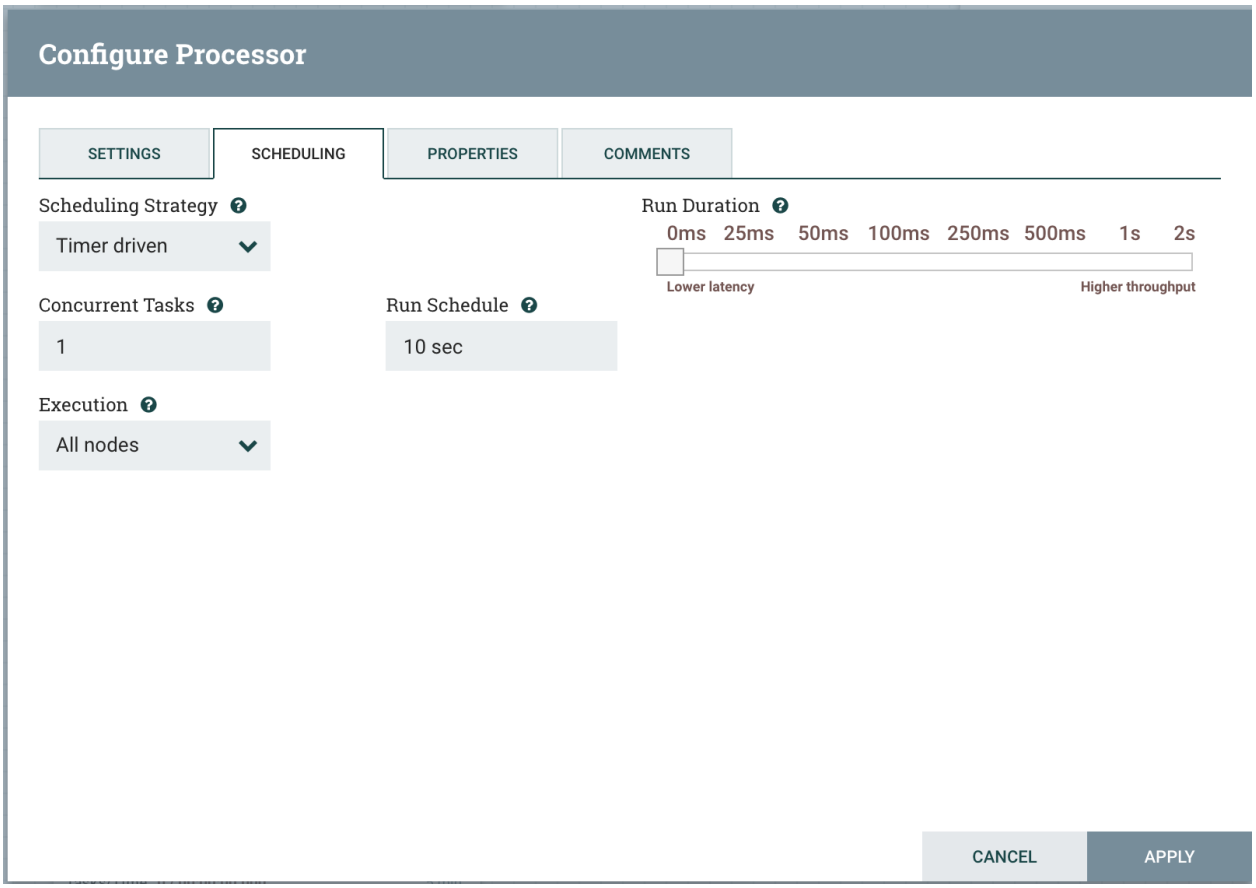


	sensorid	sensorvalue	sensortimestamp
1	3929	123	2019-01-14 10:07:17.052000000
2	7057	227	2019-01-14 10:07:07.050000000
3	8891	204	2019-01-14 10:06:57.045000000
4	3709	93	2019-01-14 10:06:36.996000000
5	6546	121	2019-01-14 10:06:47.043000000

Now If you want you can run all the components of the two flows in parallel; once done then stop the two NiFi pipelines with the **Stop** icon.

Hint

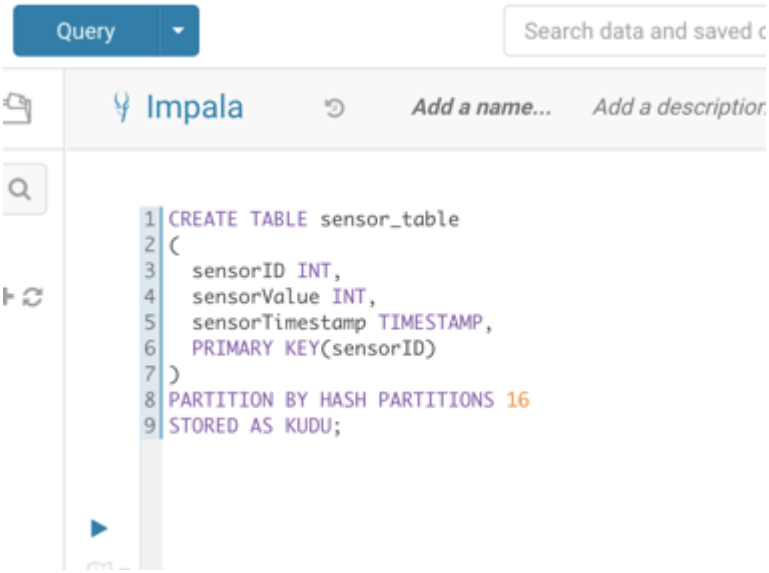
In order to generate high volume of the random sensor readings, change the **Run Schedule** field from 10 sec to a smaller value (the value of 0 sec means that the Processor should run as often as possible) or increase the number of **Concurrent Tasks**; in the **SCHEDULING** tab of the **GenerateFlowFile** component:


Configure Processor
SETTINGS
SCHEDULING
PROPERTIES
COMMENTS
Scheduling Strategy
Timer driven
Concurrent Tasks
1
Run Schedule
10 sec
Execution
All nodes
Run Duration
0ms 25ms 50ms 100ms 250ms 500ms 1s 2s
Lower latency
Higher throughput
CANCEL
APPLY

Lab 7: Improve Kudu table partitioning strategies

Partition strategy improvement #1

In our sample example we have used a simple Kudu table with a simple partitioning schema:



This schema does not take into consideration the timestamp of the reading, something that you want to consider in order to improve the performance of your solution. In this exercise we are now going to introduce a new partitioning schema considering also the timestamp field.

In Hue create a new Kudu table as following **but replace the dates in the partitions section starting with the day of today**.

```
CREATE TABLE improve_sensor_table
(
 sensorID INT,
 sensorTimestamp TIMESTAMP,
 sensorValue INT,
 PRIMARY KEY (sensorID, sensorTimestamp)
)
PARTITION BY HASH PARTITIONS 4,
RANGE (sensorTimestamp)
(
 PARTITION ('2019-02-01 00:00:00') <= VALUES < ('2019-02-02 00:00:00'),
 PARTITION ('2019-02-02 00:00:00') <= VALUES < ('2019-02-03 00:00:00')
)
STORED AS KUDU;
```

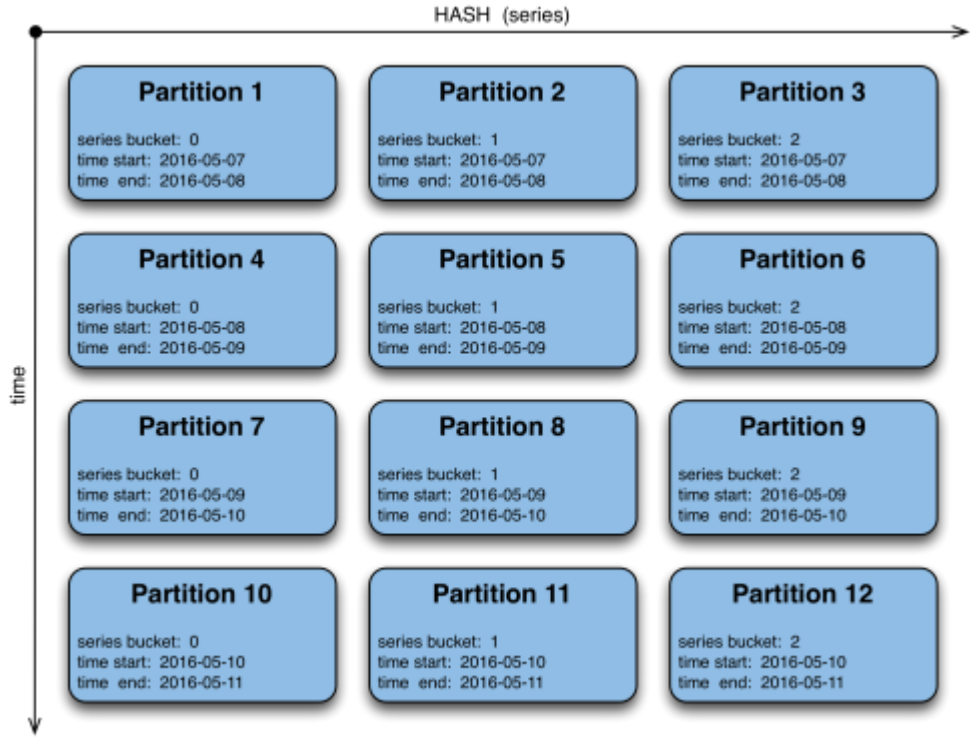
```
16
17 CREATE TABLE improve_sensor_table
18 (
19  sensorID INT,
20  sensorTimestamp TIMESTAMP,
21  sensorValue INT,
22  PRIMARY KEY (sensorID, sensorTimestamp)
23 )
24 PARTITION BY HASH PARTITIONS 4,
25 RANGE (sensorTimestamp)
26 (
27  PARTITION ('2018-10-08 00:00:00') <= VALUES < ('2018-10-09 00:00:00'),
28  PARTITION ('2018-10-09 00:00:00') <= VALUES < ('2018-10-10 00:00:00')
29 )
30 STORED AS KUDU;
31
```

```
✔ Success.

Query History    Saved Queries

un minuto fa  ✔   CREATE TABLE improve_sensor_table ( sensorID INT, sensorTimestamp TIMESTAMP, sensorValue INT,
                  PRIMARY KEY (sensorID, sensorTimestamp) ) PARTITION BY HASH PARTITIONS 4, RANGE (sensorTimestamp) (
                  PARTITION ('2018-10-08 00:00:00') <= VALUES < ('2018-10-09 00:00:00'),
                  PARTITION ('2018-10-09 00:00:00') <= VALUES < ('2018-10-10 00:00:00') ) STORED AS KUDU
```

Here we are using a nested partitioning strategy, with the hash partitioning for the sensorID and with a range partitioning for the timestamp. Conceptually it looks like this:



where we would have a dedicated partition for each day. In Kudu each partition is managed by a tablet. In a real project you would have more days and not just a couple as in our example. You cannot have more than 60 tablets during the creation of a table, so you have to add the additional

partitions after the creation of the table. Let's add new partition with the following SQL statement (**remember always to change the dates in the partition**)

```
ALTER TABLE improve_sensor_table ADD RANGE PARTITION ('2019-02-03 00:00:00') <= VALUES <
('2019-02-04 00:00:00');
```

Now that we have created a new table and added additional partitions, let's switch to the Kudu Master web interface to check the new table and partitions.

HINT – port 8051 is not yet open. Use your Azure portal to open that port.

Go to the url <your public Ip>:**8051**; then click on **Tables**:

Click on the **improve_sensor_table**:

Schema

Column	ID	Type	Encoding	Compression	Read default	Write default
sensorid	0	int32 NOT NULL	AUTO_ENCODING	DEFAULT_COMPRESSION	-	-
sensortimestamp	1	unixtime_micros NOT NULL	AUTO_ENCODING	DEFAULT_COMPRESSION	-	-
sensorvalue	2	int32 NULLABLE	AUTO_ENCODING	DEFAULT_COMPRESSION	-	-

Partition Schema

```
HASH (sensorid, sensortimestamp) PARTITIONS 4,
RANGE (sensortimestamp) (
    PARTITION 2018-09-18T00:00:00.000000Z <= VALUES < 2018-09-19T00:00:00.000000Z,
    PARTITION 2018-09-19T00:00:00.000000Z <= VALUES < 2018-09-20T00:00:00.000000Z,
    PARTITION 2018-09-20T00:00:00.000000Z <= VALUES < 2018-09-21T00:00:00.000000Z
)
```

Tablets

Summary

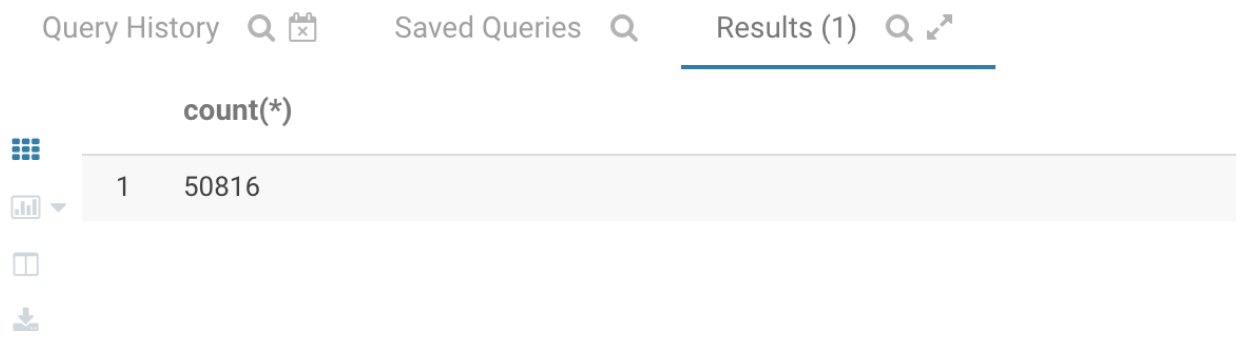
State	Count	Percentage
RUNNING	12	100.00

Where the number of Tablets is 12 = # Hash partition x # time Range

Go back to the NiFi ingest pipeline and change the table name with this new table that you have just created and run again the two flows (the ingest and the Connected Things).

Check in Hue and note (if you have left the flows running for a while - look the Hint in previous section) that now we have more than 10,000 rows because now the key is not just sensorID but sensorID and timestamp:

```
SELECT count(*) FROM improve_sensor_table;
```




```
SELECT * FROM improve_sensor_table WHERE sensorid=10000;
```

```
43 SELECT * FROM test_sensor_table WHERE sensorid=10000;
```

```
Query d5432b189214f0fa:83da202a00000000 100% Complete (12 out of 12)
```

d5432b189214f0fa:83da20

Query History | Saved Queries | Results (6)

	sensorid	sensortimestamp	sensorvalue
1	10000	2018-09-18 13:30:03.097000000	73
2	10000	2018-09-18 13:30:34.932000000	178
3	10000	2018-09-18 13:30:36.973000000	64
4	10000	2018-09-18 13:30:43.138000000	36
5	10000	2018-09-18 13:30:44.155000000	212
6	10000	2018-09-18 13:30:23.659000000	179

Stop the two NiFi pipelines.

Partition strategy improvement #2

Tablets store the rows in primary key order, so if we put the timestamp at the front of the primary key, then the tablets are mostly being appended to the end of, rather than inserted all through the middle. The reason this is mostly an append is because new rows have a higher timestamp than most of the rows that already exist.
Appends operation are better because Kudu does not need to spend time compacting new records into existing tablets row sets, and because Kudu does not need to cache metadata about every part of every tablet (if this cache starts thrashing then write performance will crater).
So, go back to Hue interface and create a new table with a better partition strategy for dealing with time-series data like those in our case (remember always to change the dates in the partitions):

```
CREATE TABLE improve2_sensor_table
(
 sensorTimestamp TIMESTAMP,
 sensorID INT,
 sensorValue INT,
 PRIMARY KEY (sensorTimestamp, sensorID)
)
PARTITION BY HASH PARTITIONS 4,
RANGE (sensorTimestamp)
(
 PARTITION ('2019-02-01 00:00:00') <= VALUES < ('2019-02-02 00:00:00'),
 PARTITION ('2019-02-02 00:00:00') <= VALUES < ('2019-02-03 00:00:00')
)
STORED AS KUDU;
```

```
CREATE TABLE improve2_sensor_table
(
 sensorID INT,
 sensorTimestamp TIMESTAMP,
 sensorValue INT,
 PRIMARY KEY (sensorTimestamp, sensorID)
)
PARTITION BY HASH PARTITIONS 4,
RANGE (sensorTimestamp)
(
 PARTITION ('2018-10-08 00:00:00') <= VALUES < ('2018-10-09 00:00:00'),
 PARTITION ('2018-10-09 00:00:00') <= VALUES < ('2018-10-10 00:00:00')
)
STORED AS KUDU;

Success.

Query History   Saved Queries

alcuni secondi fa   CREATE TABLE improve2_sensor_table ( sensorID INT, sensorTimestamp TIMESTAMP, sensorValue INT,
                    PRIMARY KEY (sensorTimestamp, sensorID) ) PARTITION BY HASH PARTITIONS 4, RANGE (sensorTimestamp) (
                    PARTITION ('2018-10-08 00:00:00') <= VALUES < ('2018-10-09 00:00:00'),
                    PARTITION ('2018-10-09 00:00:00') <= VALUES < ('2018-10-10 00:00:00') ) STORED AS KUDU
```

Go back to the NiFi ingest pipeline and change the table name with this new table that you have just created, run again the two pipelines (the ingest and the Connected Things).

Once you have and verify in Hue that this new table is being populated, then you can stop the two StreamSets pipelines.

Simple Queries

Let's understand more the values ingested in Kudu. First we need to identify the sensor that has generated more values, in order to do to type in Hue the following query:

```
select count(*) as Y, sensorid
from improve2_sensor_table
group by sensorid
order by Y DESC
limit 10;
```

```
59 select count(*) as Y, sensorid
60 from improve2_sensor_table
61 group by sensorid
62 order by Y DESC
63 limit 10;
64
```

```
Query ef4f7849b5dc04b4:c5d0de2b00000000: 0% Complete (0 out of 8)
Query ef4f7849b5dc04b4:c5d0de2b00000000 100% Complete (8 out of 8)
Query ef4f7849b5dc04b4:c5d0de2b00000000 100% Complete (8 out of 8)
```

ef4f7849b5dc04b4:c5d0de2b00000000

Query History | Saved Queries | Results (10)

	y	sensorid
1	12	2794
2	12	3724
3	12	817
4	11	2979
5	11	1169
6	11	4059
7	11	7628
8	11	3396
9	10	9360
10	10	2569

In my case the sensor with ID 2794 has 12 reading. Let's find out the min, max and average values for this sensor:

```
select min(sensorvalue) from improve2_sensor_table where sensorid=2794;
```

```
68 select min(sensorvalue) from improve2_sensor_table where sensorid=2794;
69
```

```
Query 744a0f8717c36b08:31cefd5c00000000: 62% Complete (5 out of 8)
Query 744a0f8717c36b08:31cefd5c00000000 100% Complete (8 out of 8)
```

744a0f8717c36b08:31cefd5c00000000

Query History | Saved Queries | Results (1)

	min(sensorvalue)
1	66

```
select max(sensorvalue) from improve2_sensor_table where sensorid=2794;
```

```
70 select max(sensorvalue) from improve2_sensor_table where sensorid=2794;
71
```

```
Query 7741b899716233d0:a39bcd8600000000 100% Complete (8 out of 8)
```

7741b899716233d0:a39bcd8600000000

Query History | Saved Queries | Results (1)

	max(sensorvalue)
1	252

```
select avg(sensorvalue) from improve2_sensor_table where sensorid=2794;
```

```
71
72 select avg(sensorvalue) from improve2_sensor_table where sensorid=2794;
73
```

```
Query f445b364c5df79d1:7177c2de00000000: 62% Complete (5 out of 8)
Query f445b364c5df79d1:7177c2de00000000 100% Complete (8 out of 8)
```

f445b364c5df79d1:7177c2de00000000

Query History | Saved Queries | Results (1)

	avg(sensorvalue)
1	167.33333333333334

This concludes the IoT/Time series data and real-time streaming lab.

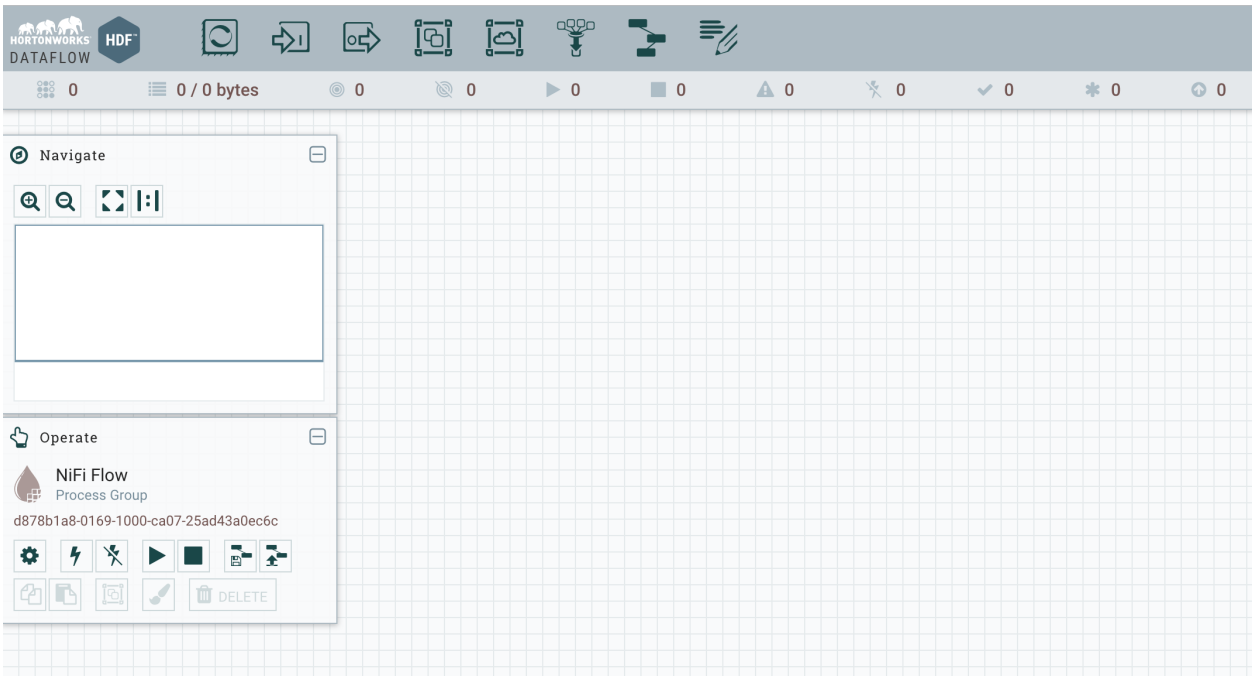
APPENDIX - NiFi solution

Here are the NiFi flows already implemented that can be imported.
Before to run them make sure you have changed the Kafka brokers and Kudu master urls to point to your cluster.

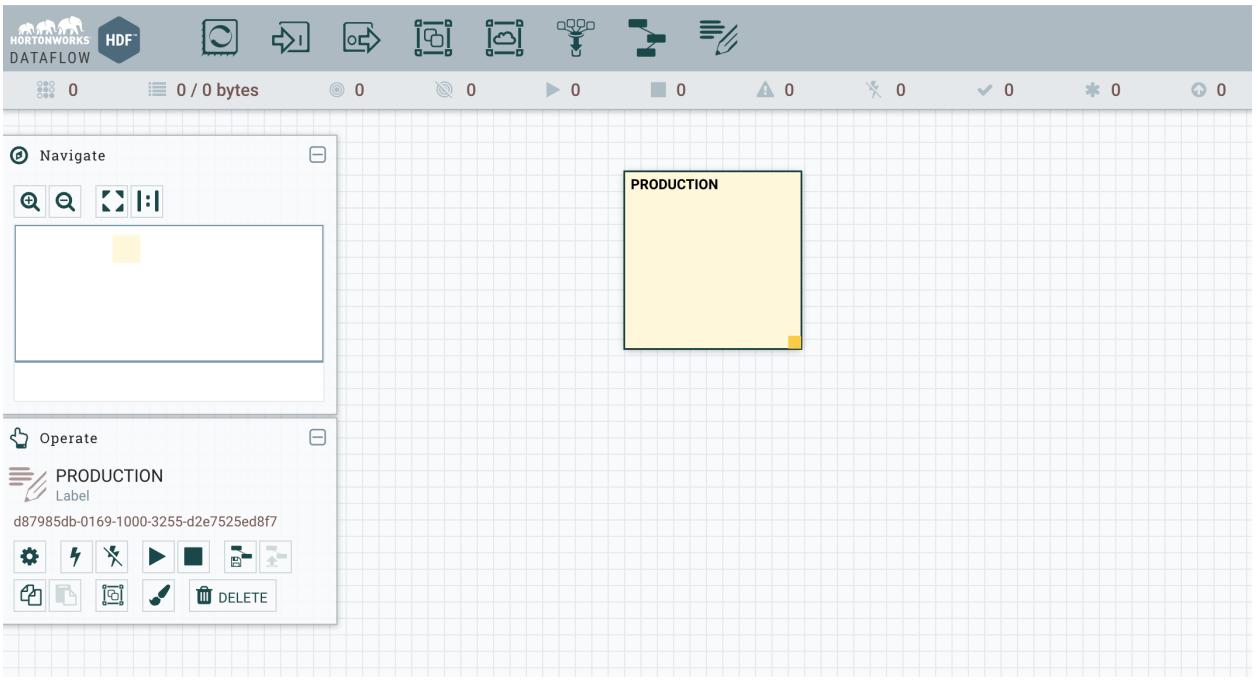
generateDataSensors
ingestDataSensors

APPENDIX - Promote a flow in Production

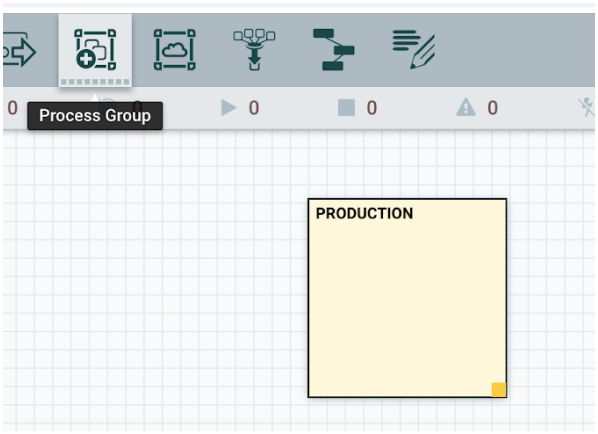
If you have a second NiFi installation to simulate the PRODUCTION environment, then connect to its Web UI:



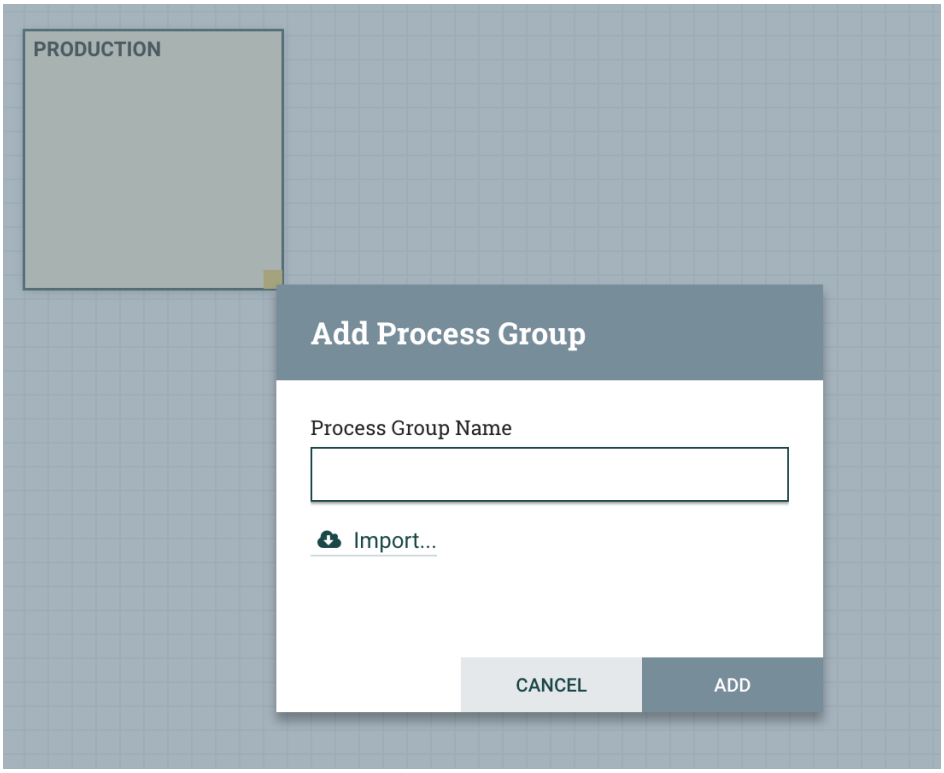
Add a label and called it PRODUCTION



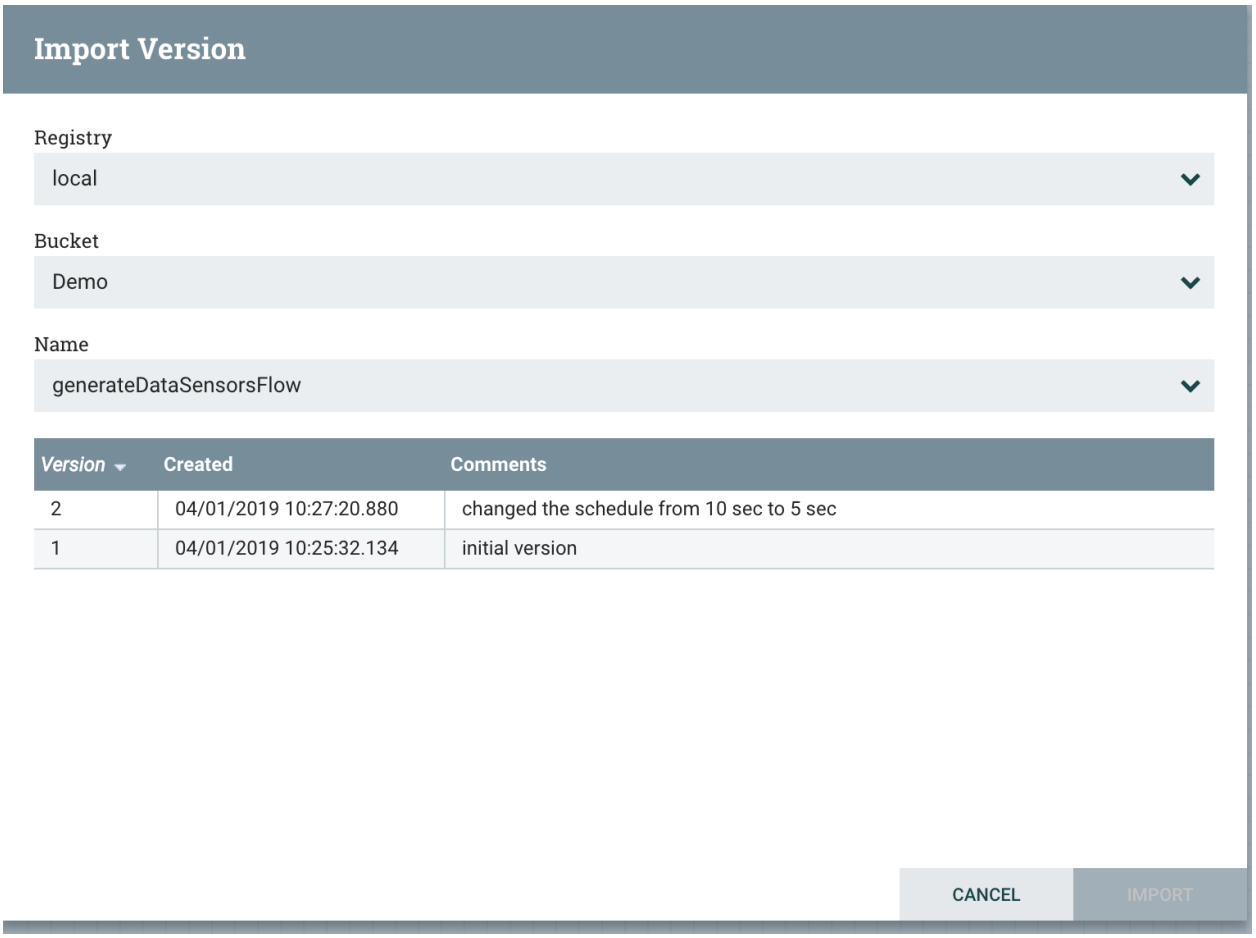
Now connect the NiFi Registry, remember how we did before?
Select the **Process Group** and …



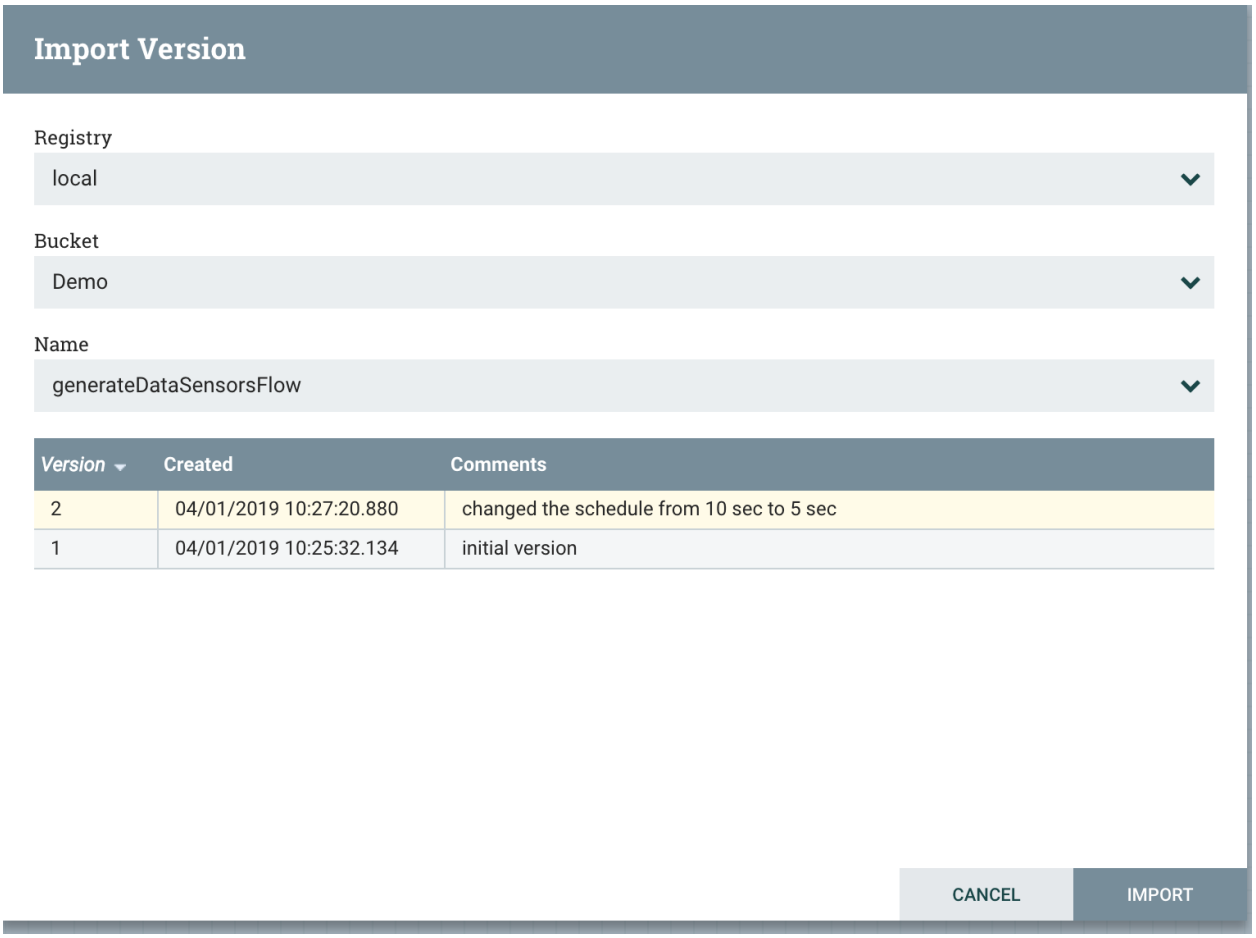
Drag&drop a Process Group to the canvas:



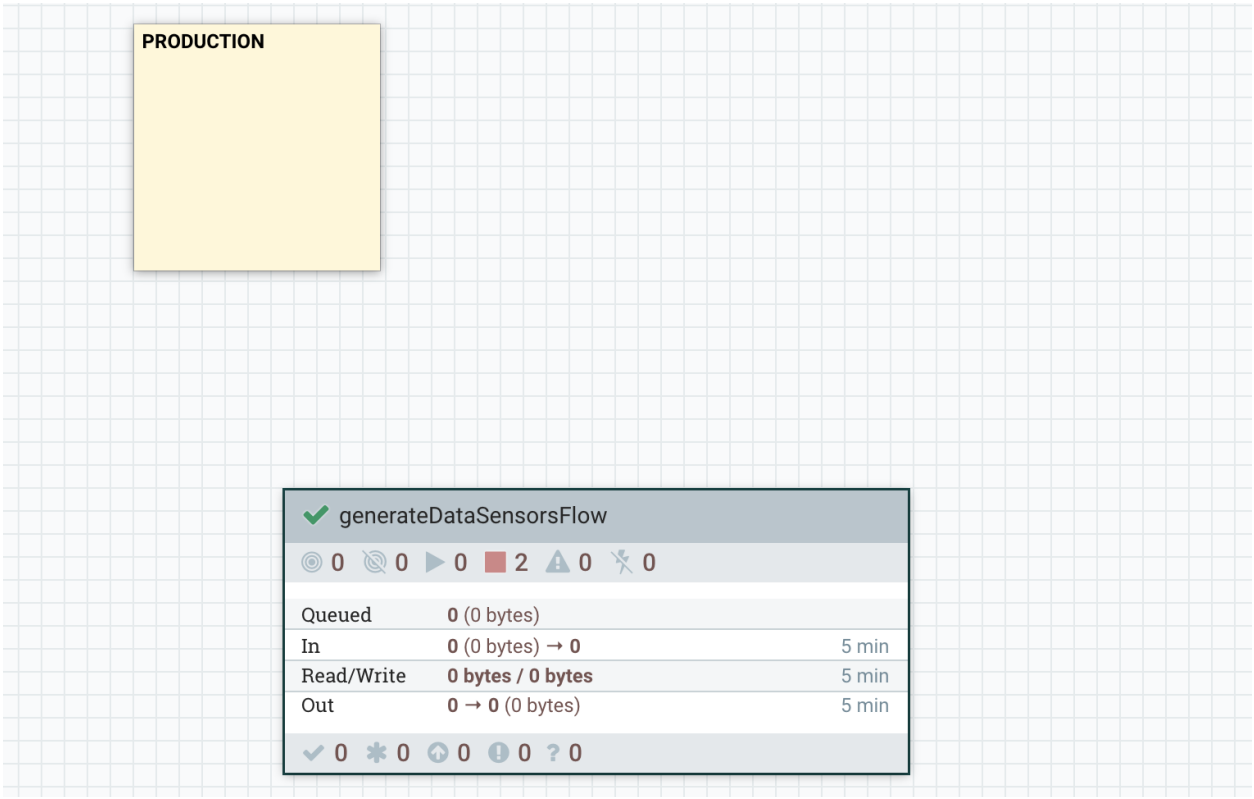
Select **Import**:



Select the version 2



And then **IMPORT**



we have an identical flow in a second NiFi instance.

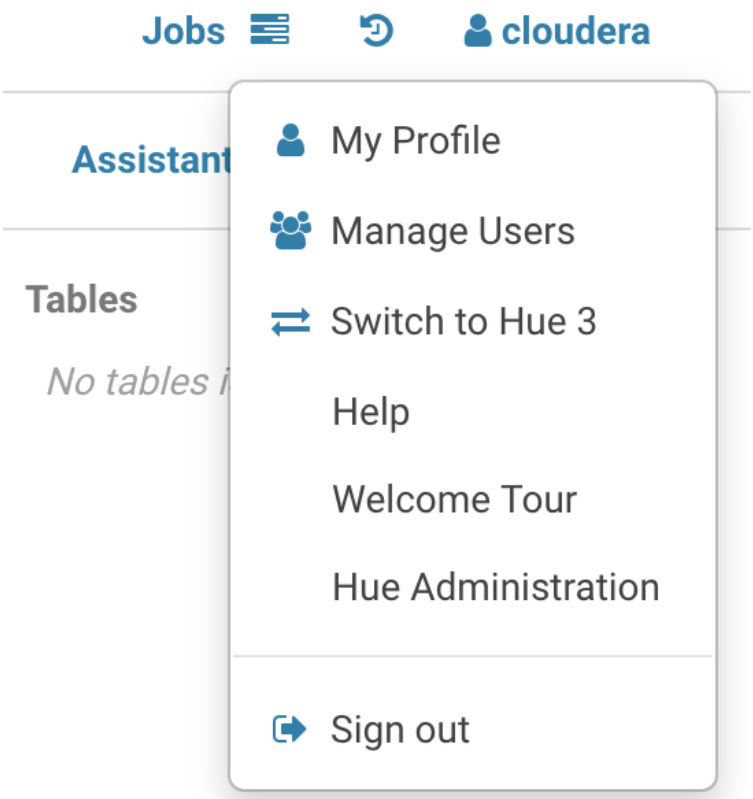
APPENDIX - Using NiFi to ingest Web logs into EDH

Getting the Web Logs file

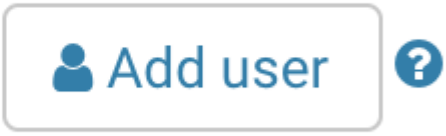
For this lab, for simplicity, we are going to upload a web logs file into HDFS and then we will configure a NiFi flow for splitting each line, reading it, extract the fields and then save them in Kudu. For real scenarios typically the external application will send each line as it is being generated to NiFi.

We are going to use the "*nifi*" user, to store and read the web logs file in HDFS, so let's create it with the HUE Web UI.

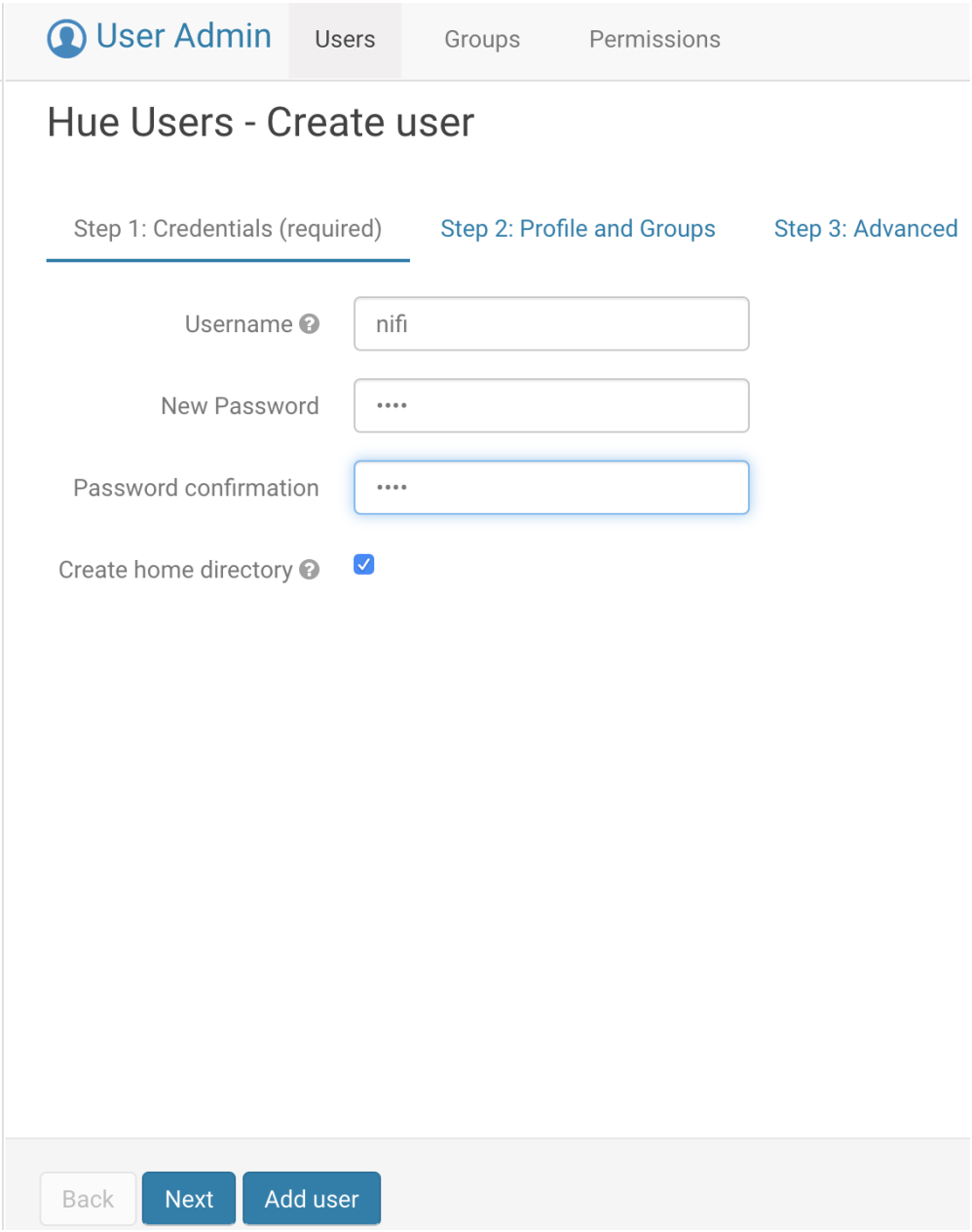
Go to **Manage Users**:


Jobs
cloudera
Assistant
Tables
No tables i
My Profile
Manage Users
Switch to Hue 3
Help
Welcome Tour
Hue Administration
Sign out

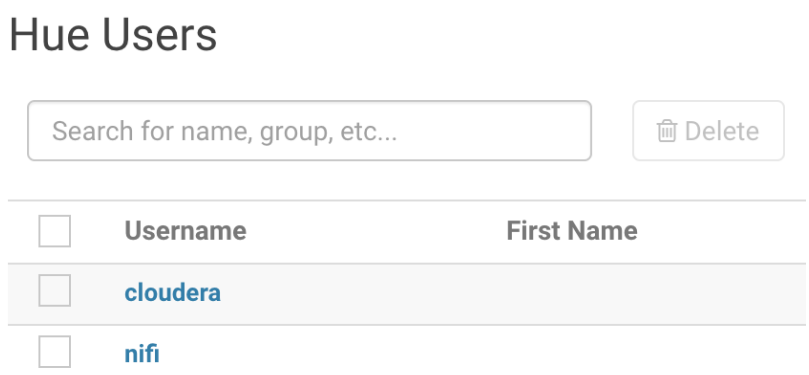
Then **Add User**



with:
Username: *nifi*
Password: *nifi*

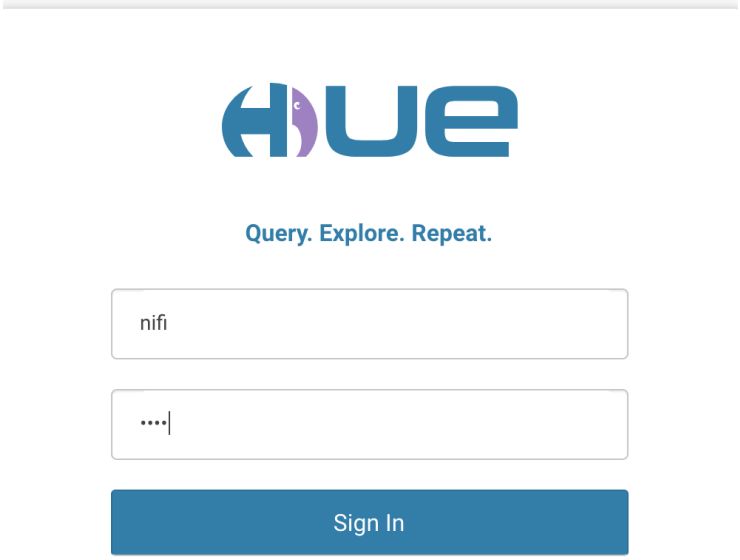


and then **Add user**.

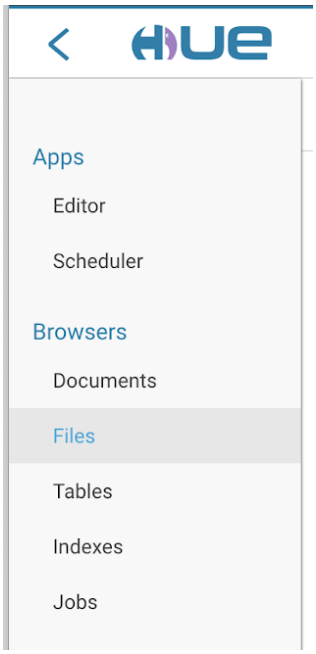


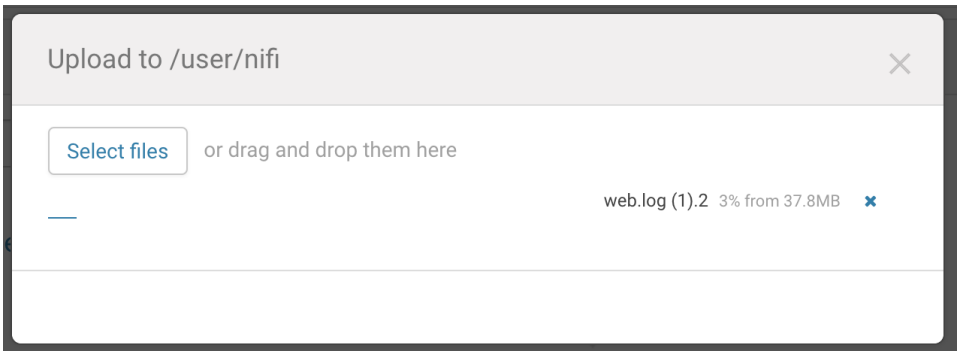
Download the following file to your laptop **web.log.2**
https://cloudera.box.com/s/52m5007eqx2asmx0l9c5tf9woazu64fh

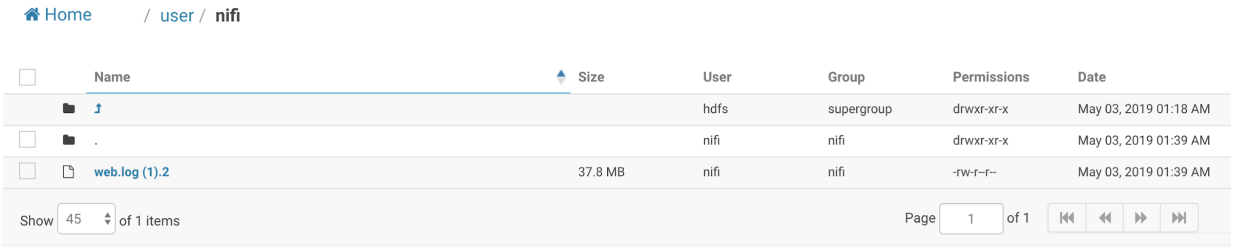
Then login to HUE with the *nifi* user



and upload the web logs file into the HDFS folder */user/nifi* using the HUE File Browser.







Creating a Web Logs table

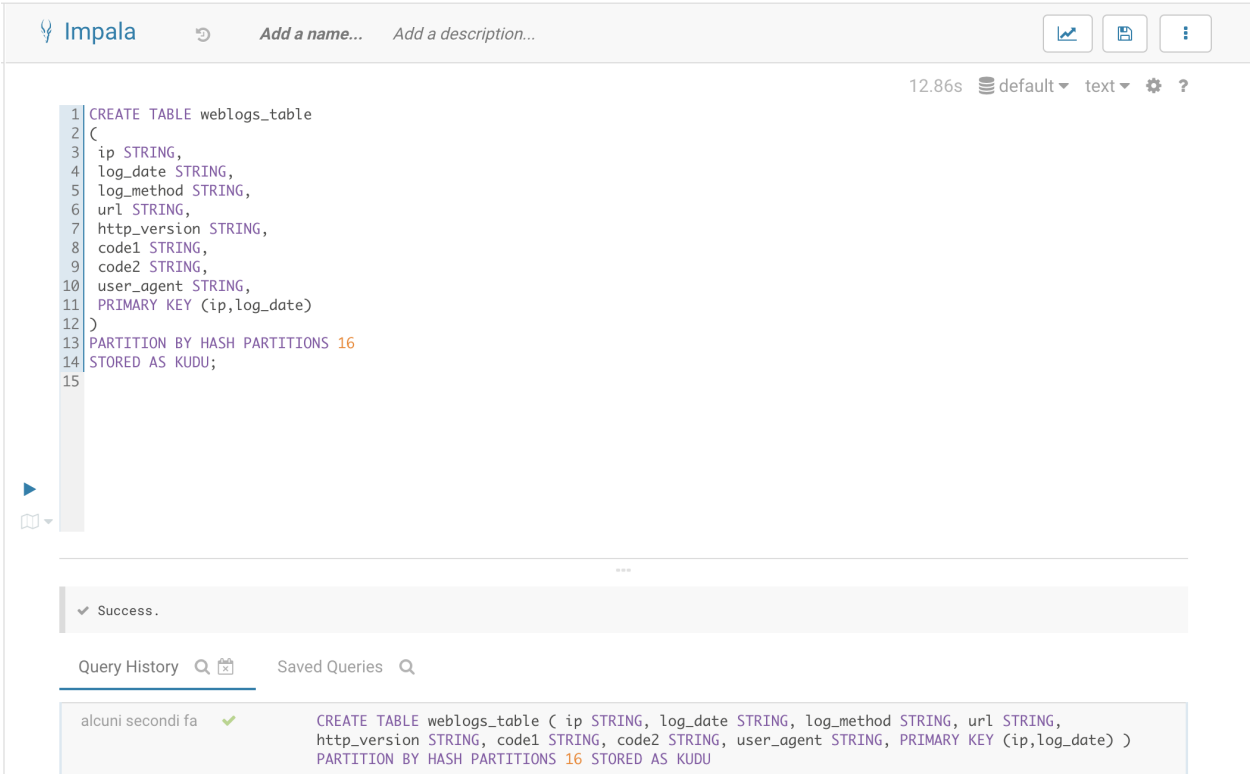
Open the Web log file to see how its lines look like:

```
79.133.215.123 - - [14/Jun/2014:10:30:13 -0400] "GET /home HTTP/1.1" 200 1671 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36"
```

For each line we are going to extract the IP, the date, the URL, the HTTP version and so on ...and store these values inside a store, that for simplicity we have decided to be kudu. So in order to do that we need to create a kudu table for this purpose.

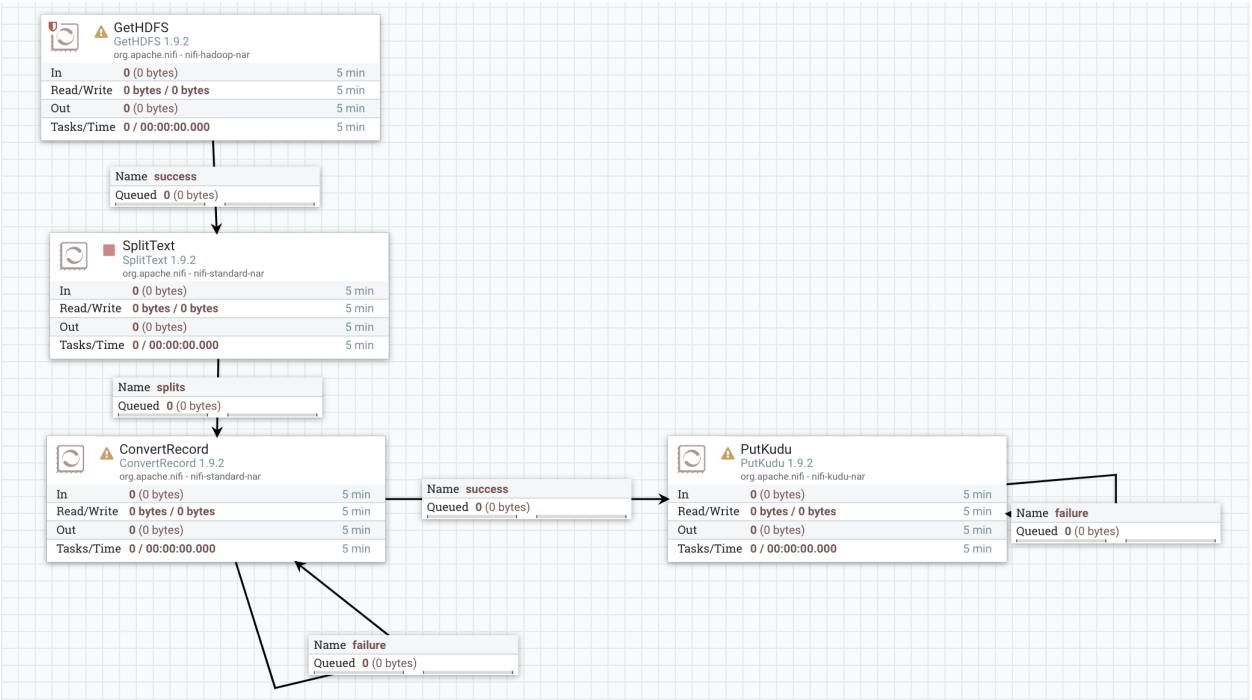
Within the Impala Query Editor type the following SQL statement:

```
CREATE TABLE weblogs_table
(
ip STRING,
log_date STRING,
log_method STRING,
url STRING,
http_version STRING,
code1 STRING,
code2 STRING,
user_agent STRING,
PRIMARY KEY (ip,log_date)
)
PARTITION BY HASH PARTITIONS 16
STORED AS KUDU;
```



Creating the NiFi Flow

Now we are ready to configure the following NiFi flow:

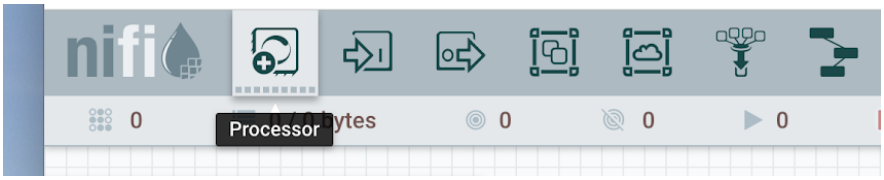


The flow should be able to do the following:

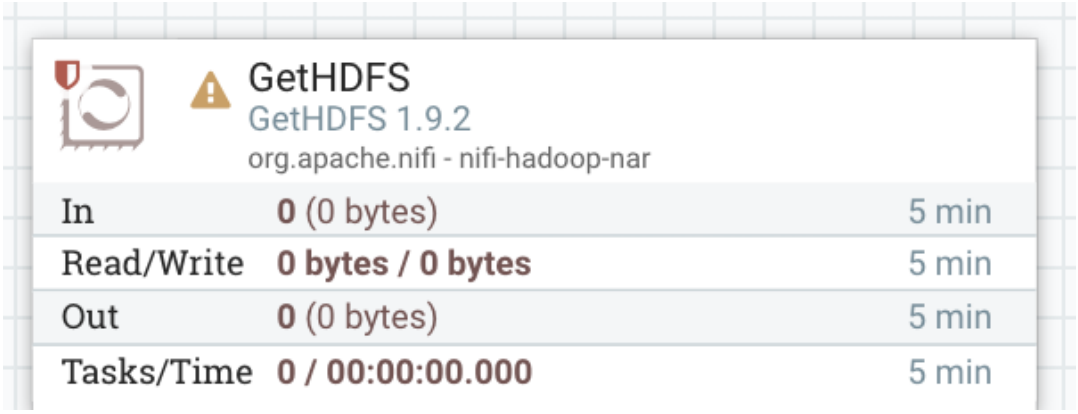
1. read the file from HDFS
2. split each line
3. extract the fields we are interested in
4. store them into kudu

Let's build it!

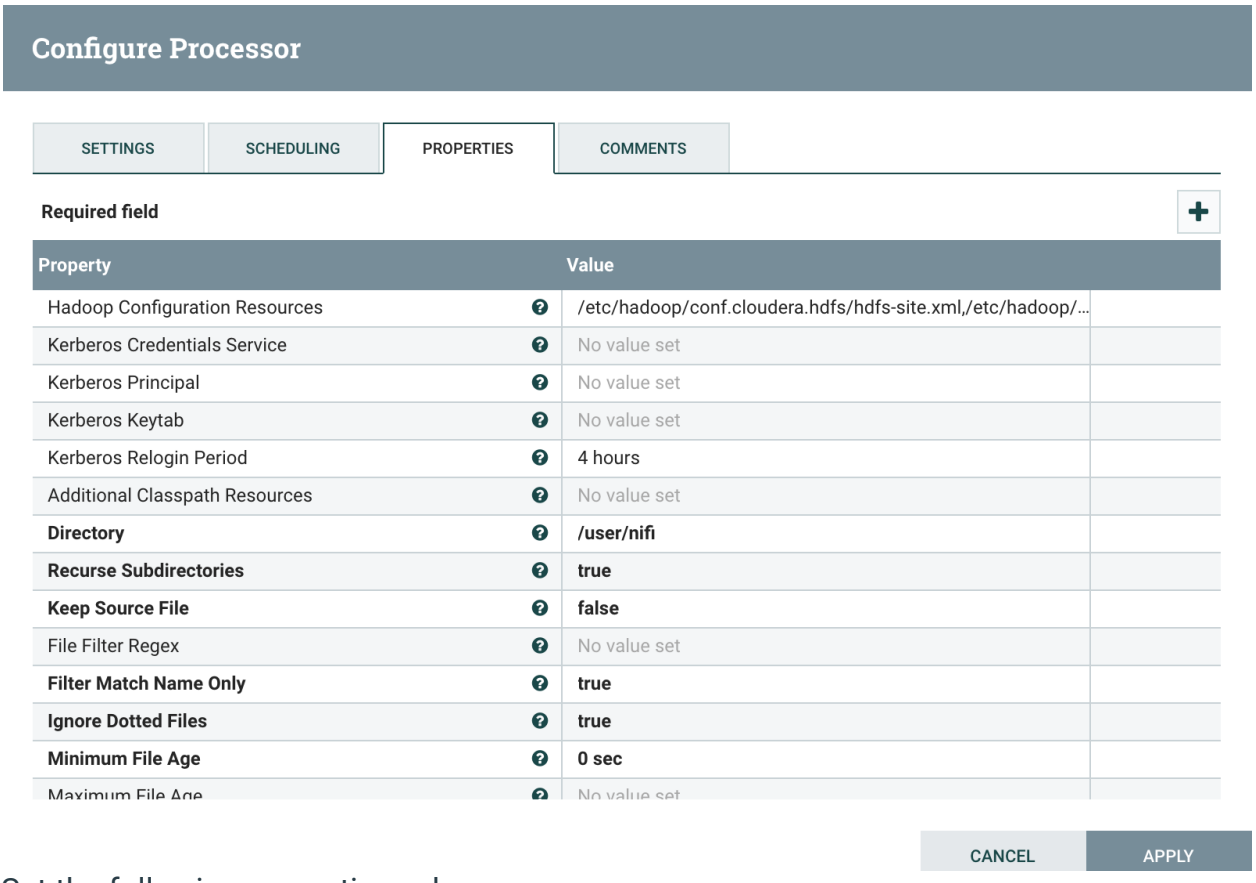
On the top-left corner, click on the **Processor** icon and drag it into the canvas:



and select **GetHDFS**:



Double click the processor and go to the **PROPERTIES** tab:

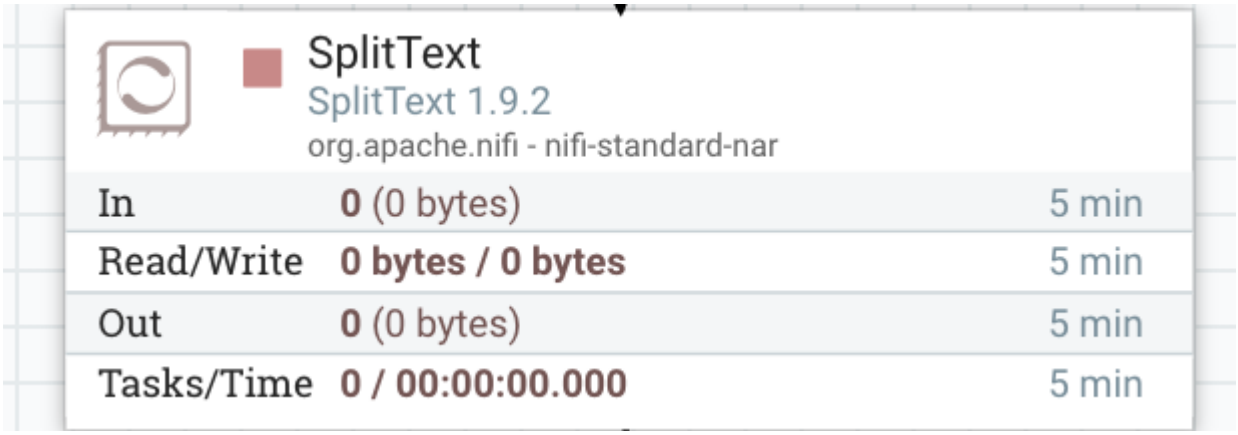


Set the following properties values:

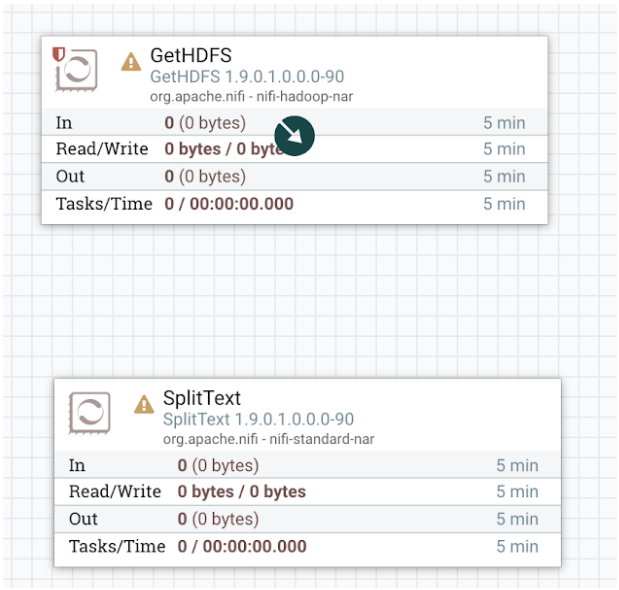
Hadoop Configuration Resources: `/etc/hadoop/conf.cloudera.hdfs/hdfs-site.xml,/etc/hadoop/conf.cloudera.hdfs/core-site.xml`

Directory: `/user/nifi`

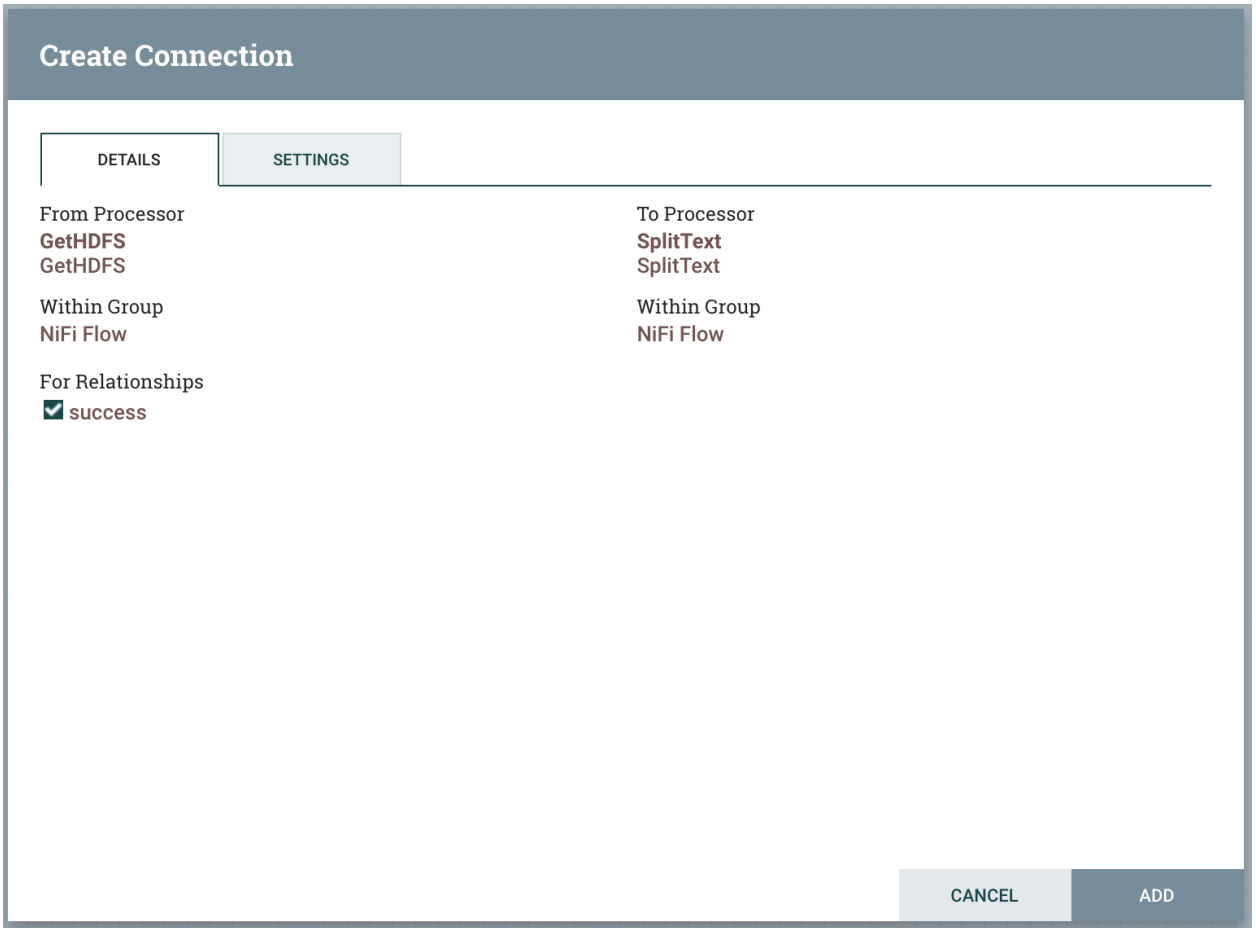
Then add another processor to the flow, the **SplitText**:



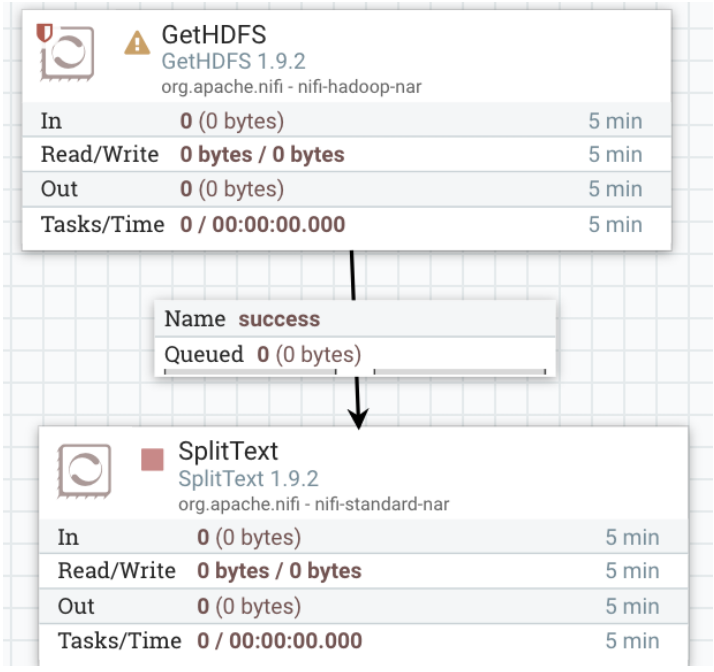
and connect the two processor. With the mouse, go over the **GetHDFS** step and you should see an arrow icon and drag it to the **SplitText**:



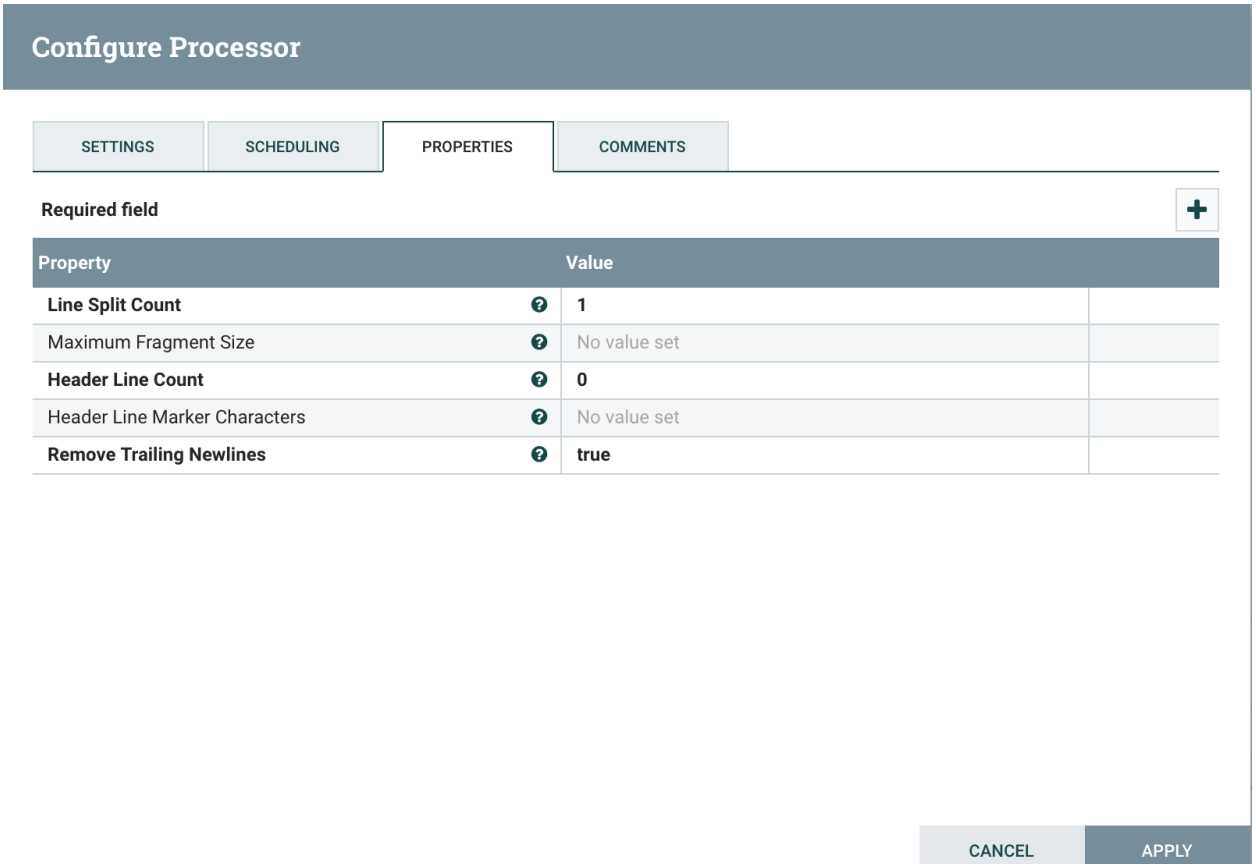
accept the default



and click **ADD**



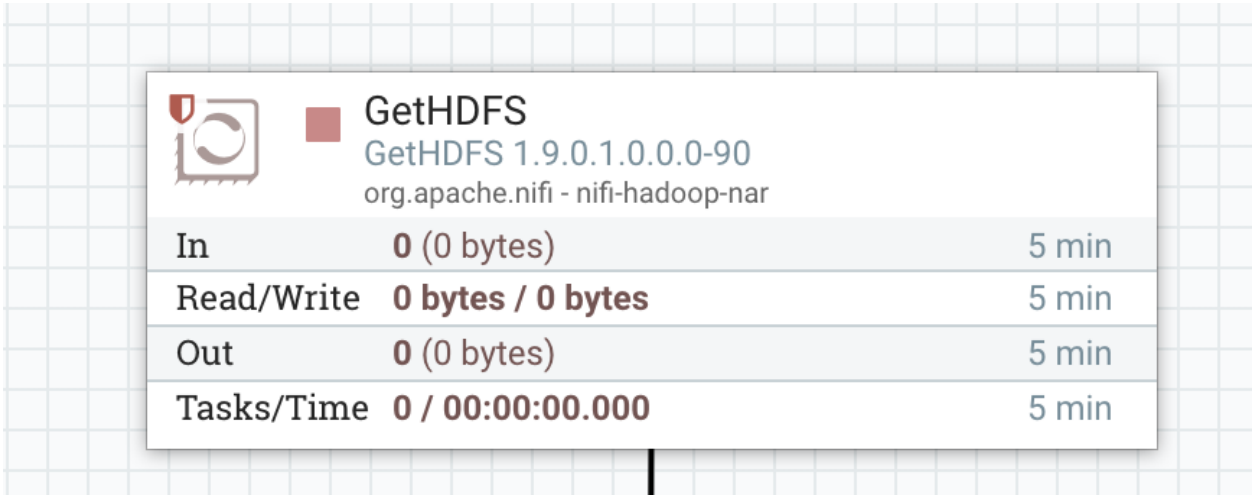
Double click on the **SplitText** and go to the **PROPERTIES** tab:



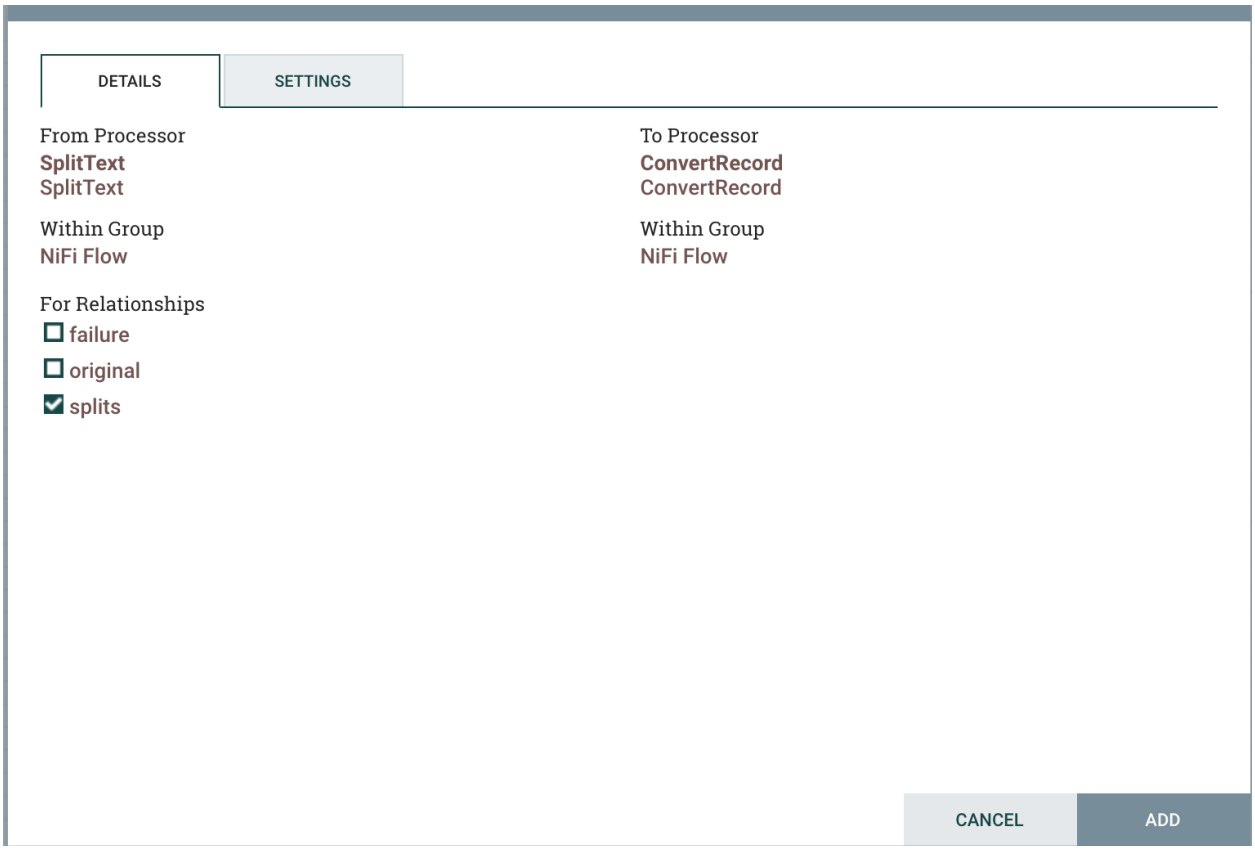
Set 1 to the **Line Split Count** and click **APPLY**.

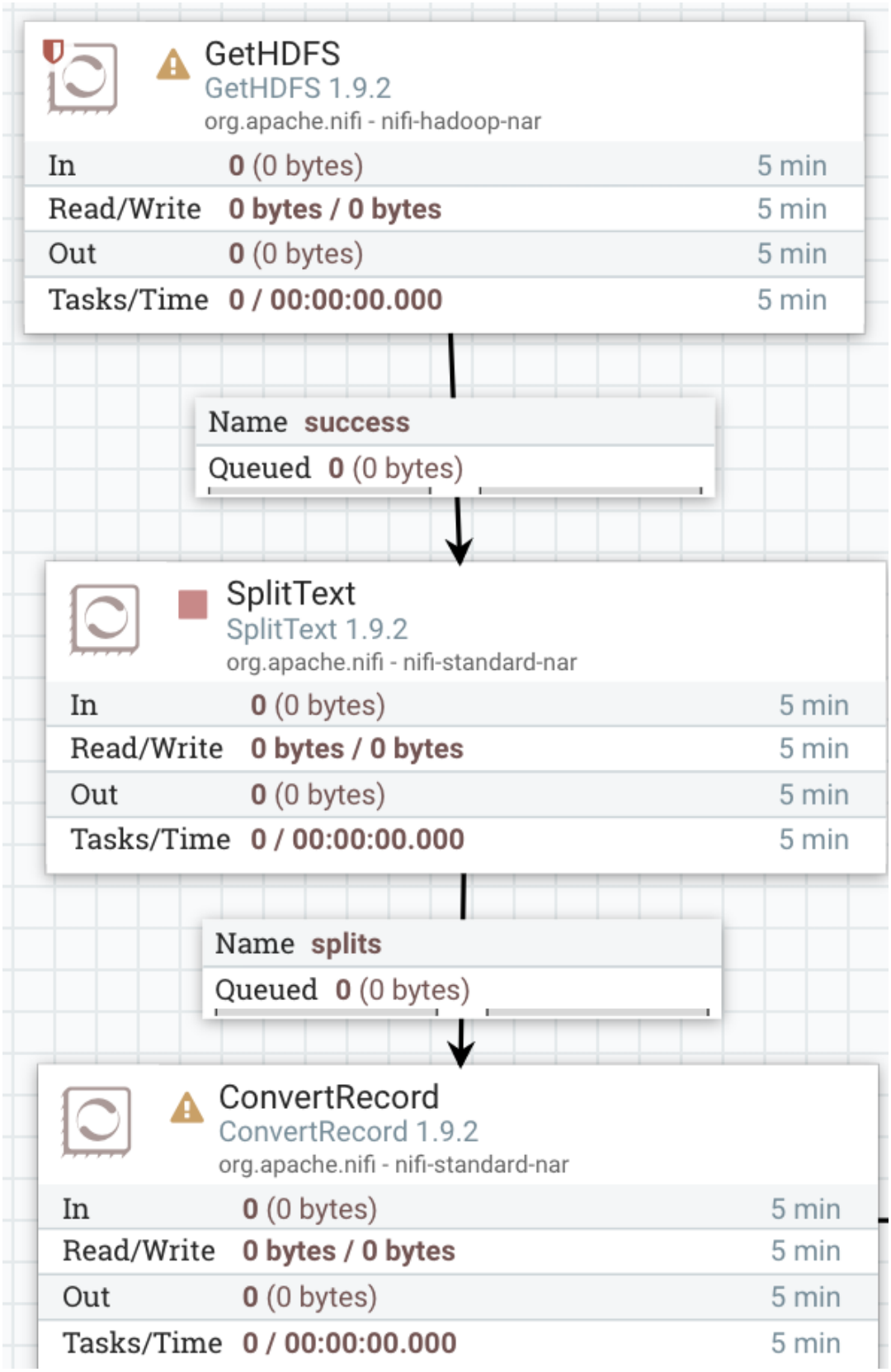
This step will extract one line at the time and pass it forward to the next step, that is going to understand and detect the fields (IP, URL, …).

After this step the icon on the left top corner of GetHDFS should be a square indicating that was able to access and get the hdfs directory:

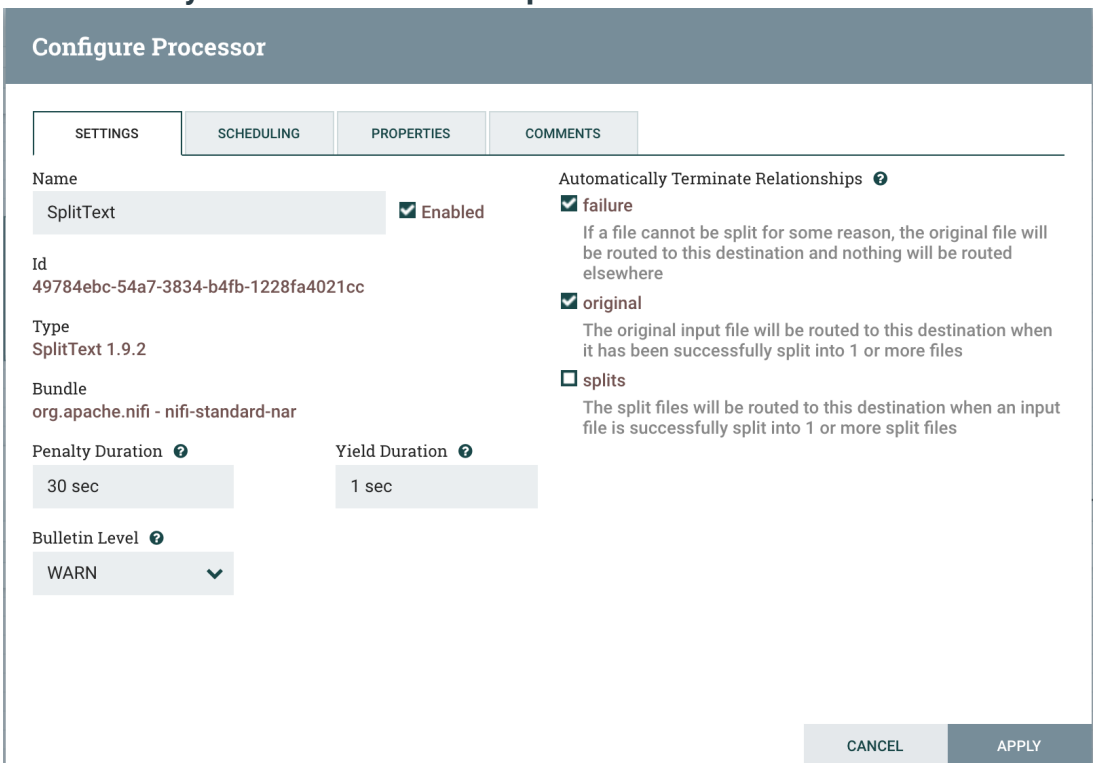


Add a **ConvertRecord** processor and connect to the **SplitText**, by setting the **splits** option in the connection details:

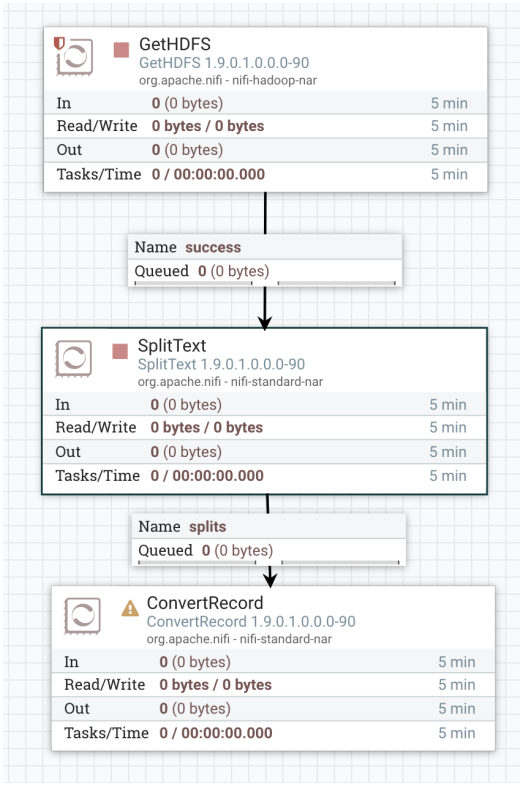



GetHDFS
GetHDFS 1.9.2
org.apache.nifi - nifi-hadoop-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
SplitText
SplitText 1.9.2
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name splits
Queued 0 (0 bytes)
ConvertRecord
ConvertRecord 1.9.2
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

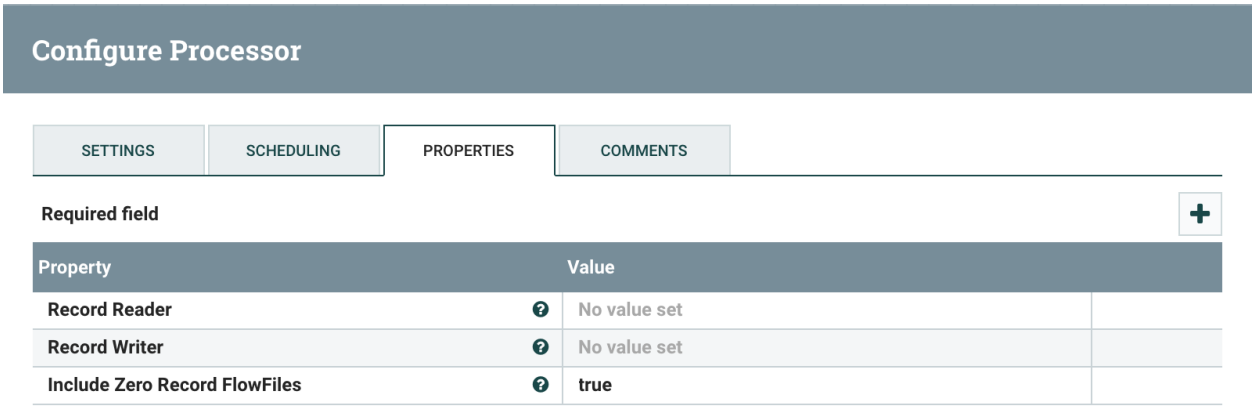
Double click on the **SplitText**, **SETTINGS** tab and check ***failure*** and ***original*** for **Automatically Terminate Relationships**



And now also the **SplitText** has a red square:



Double click on the **ConvertRecord** and go to the **PROPERTIES** tab:



The **Record Reader** property specify how to read the incoming data, while the **Record Write** how to write out the data.

Because the logs line look like the following:

```
79.133.215.123 - - [14/Jun/2014:10:30:13 -0400] "GET /home HTTP/1.1" 200 1671 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWe
bKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36"
```

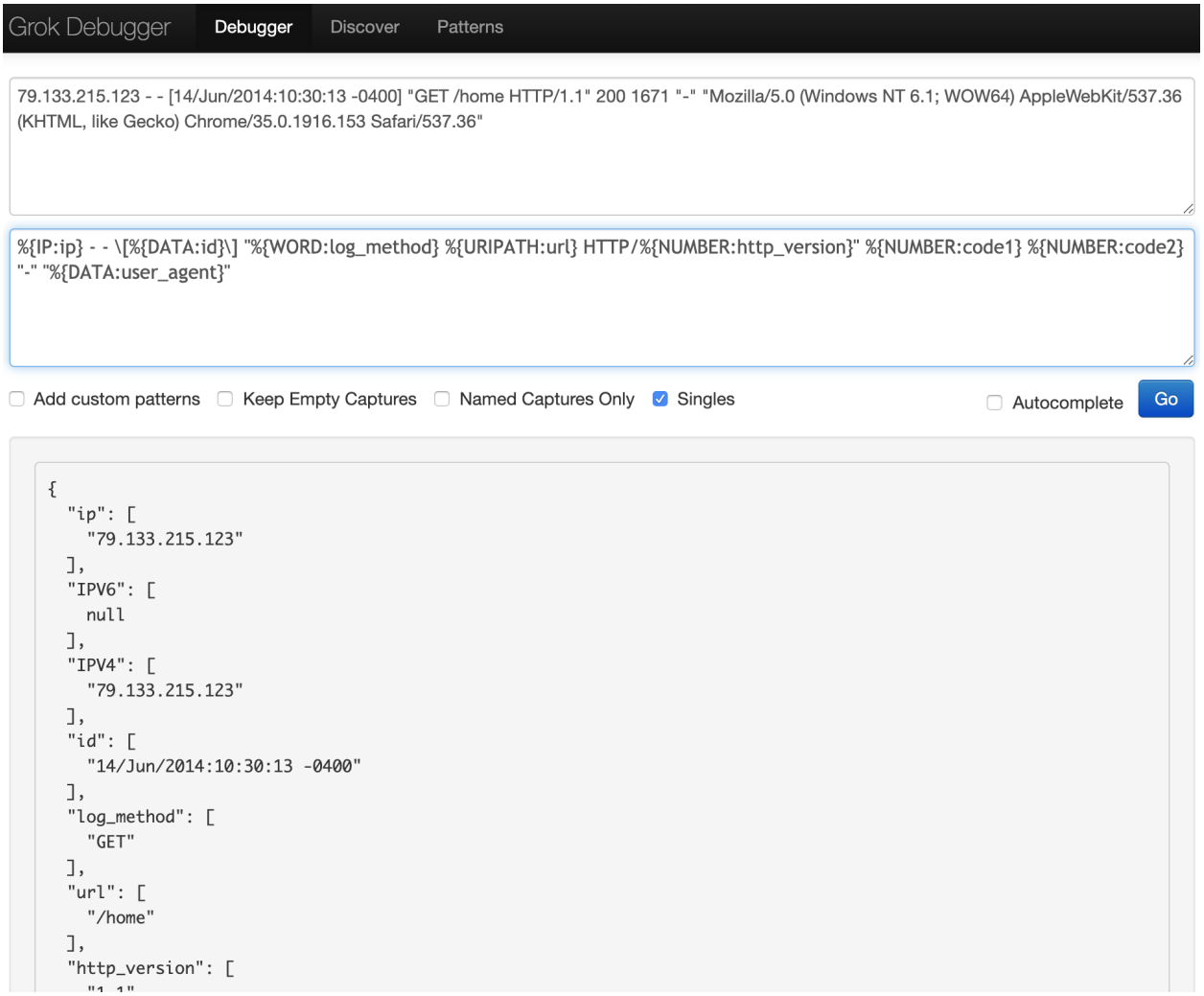
we are going to use GROK.

In order to correctly write the GROK expression, go to this GROK debugger https://grokdebug.herokuapp.com/

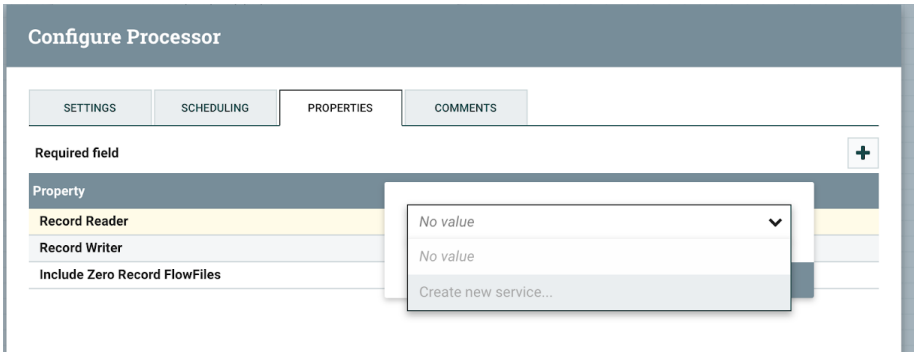
Paste the logs line and then this GROK expression:

```
%{IP:ip} - - \[%{DATA:id}\] "%{WORD:log_method} %{URIPATH:url}
HTTP/%{NUMBER:http_version}" %{NUMBER:code1} %{NUMBER:code2} "-"
"%{DATA:user_agent}"
```

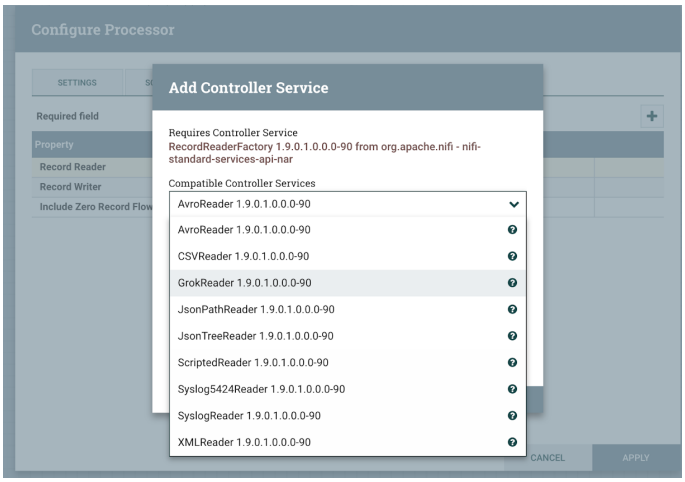
and you should see our fields extracted:

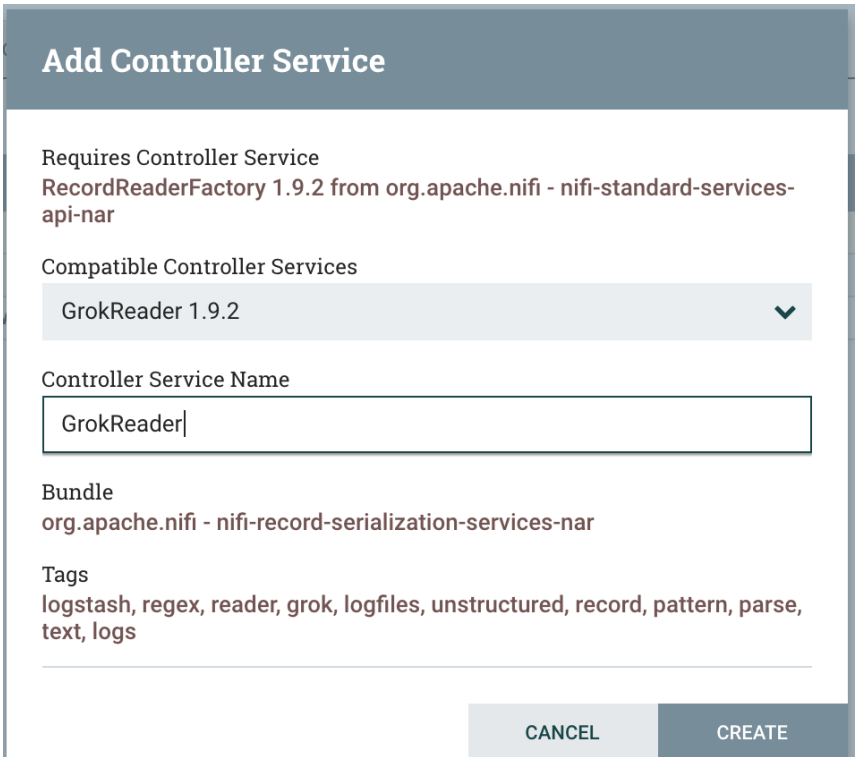


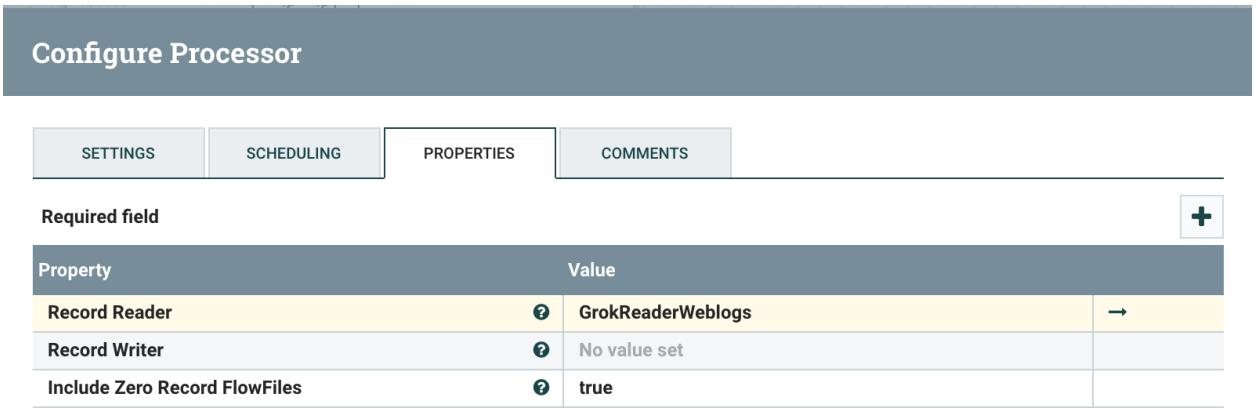
Click on the **Value** of the **Record Reader** and select **Create new service**…



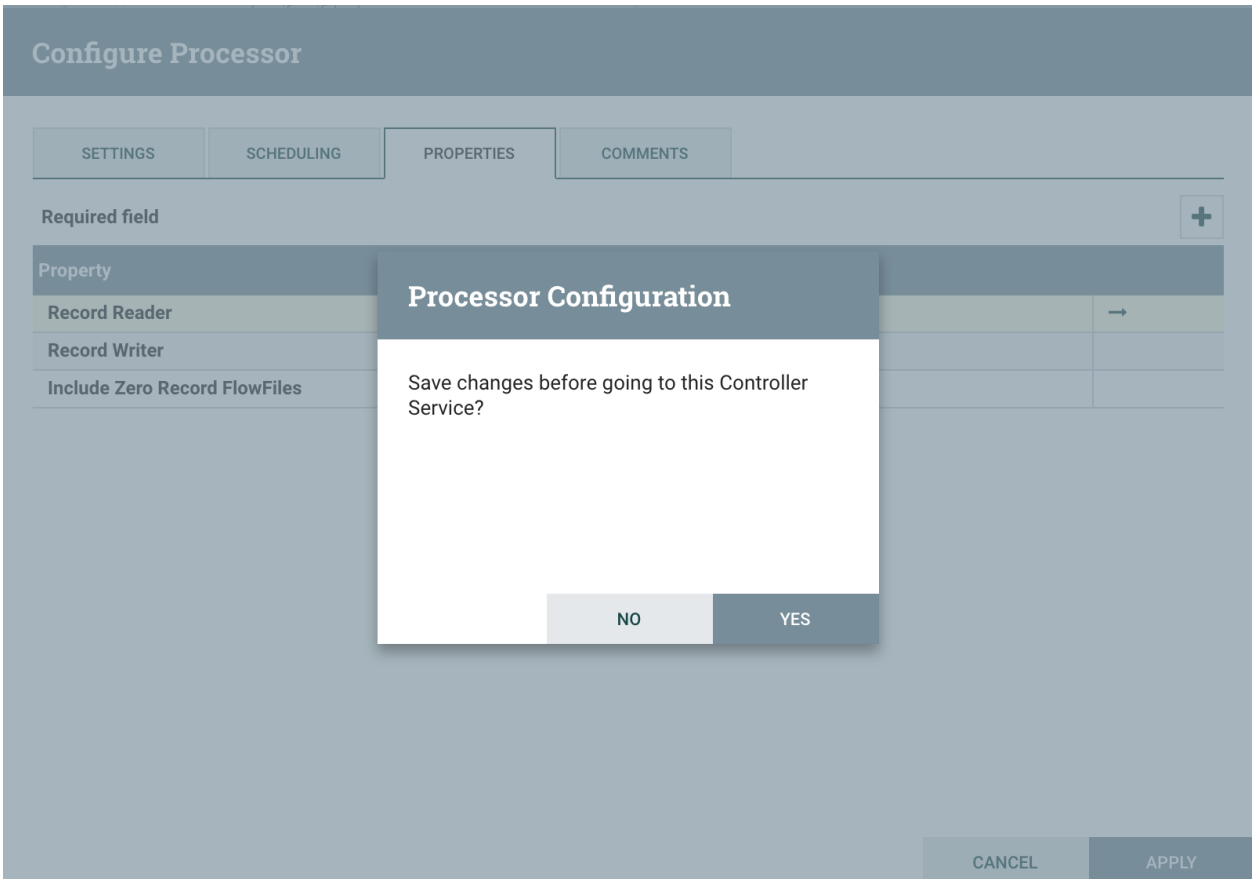
Select GrokReader and give it a name (for example GrokReaderWeblogs…)

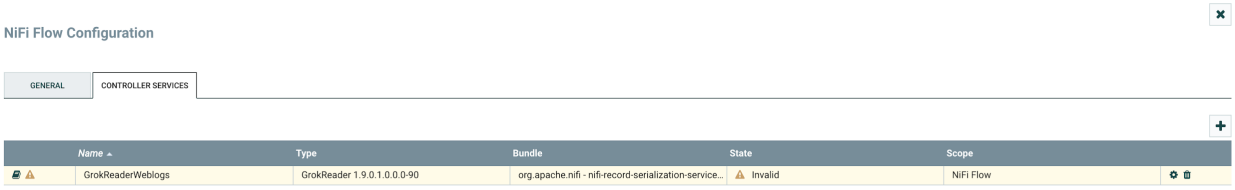




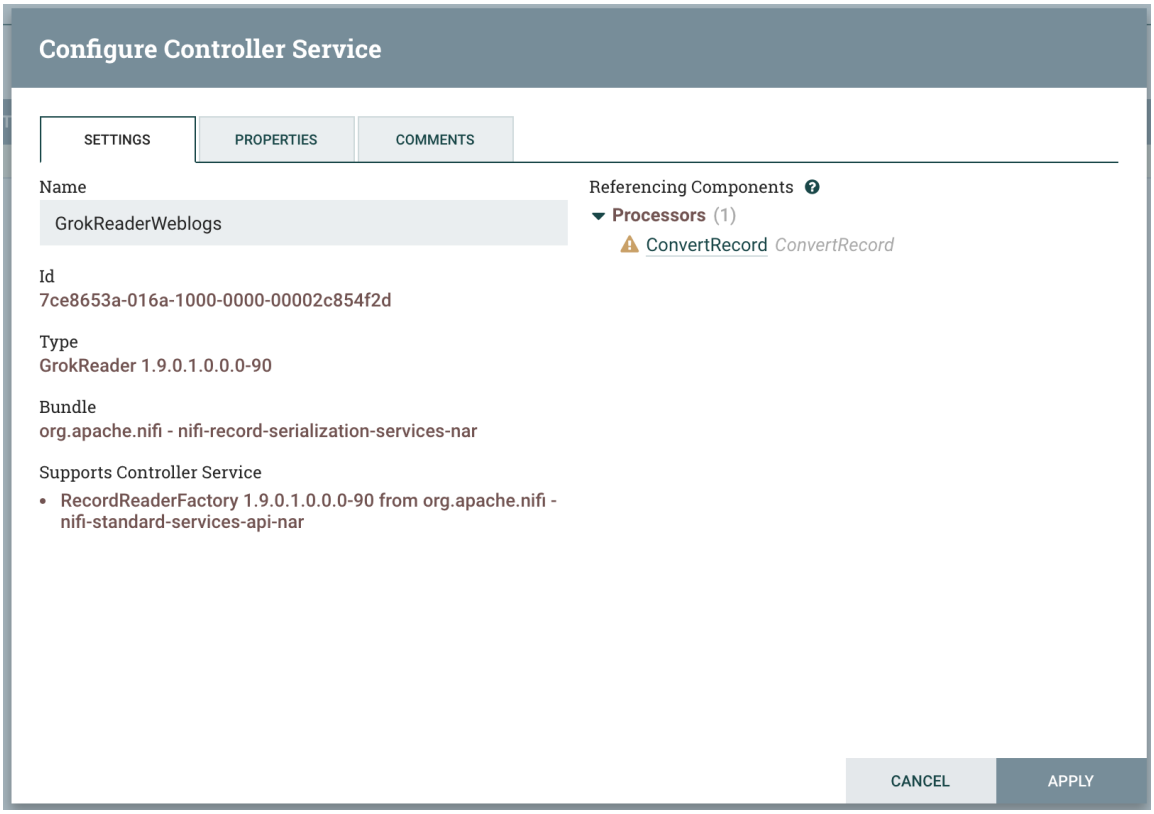


Then click on the arrow on the right, and save changes

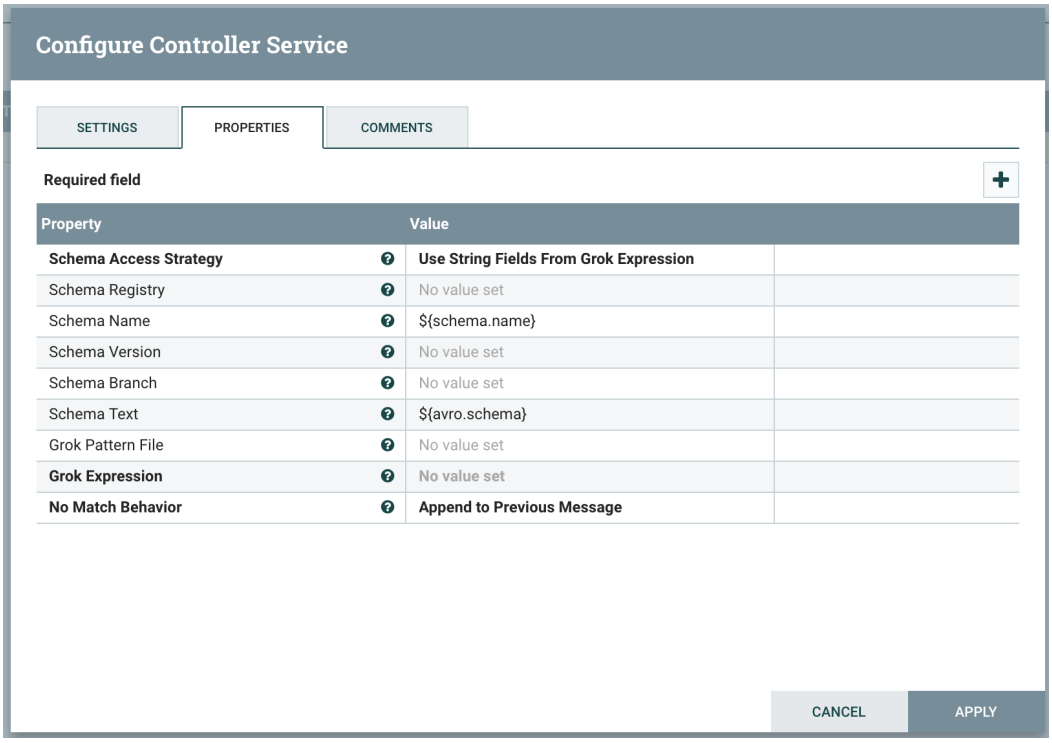

Configure Processor
SETTINGS
SCHEDULING
PROPERTIES
COMMENTS
Required field
Property
Record Reader
Record Writer
Include Zero Record FlowFiles
Processor Configuration
Save changes before going to this Controller Service?
NO
YES
CANCEL
APPLY


NiFi Flow Configuration
GENERAL
CONTROLLER SERVICES
Name
Type
Bundle
State
Scope
GrokReaderWeblogs
GrokReader 1.9.0.1.0.0.0-90
org.apache.nifi - nifi-record-serialization-service...
Invalid
NiFi Flow

and then click on the **gear icon**, for configuring it:



Go to the **PROPERTIES** tab



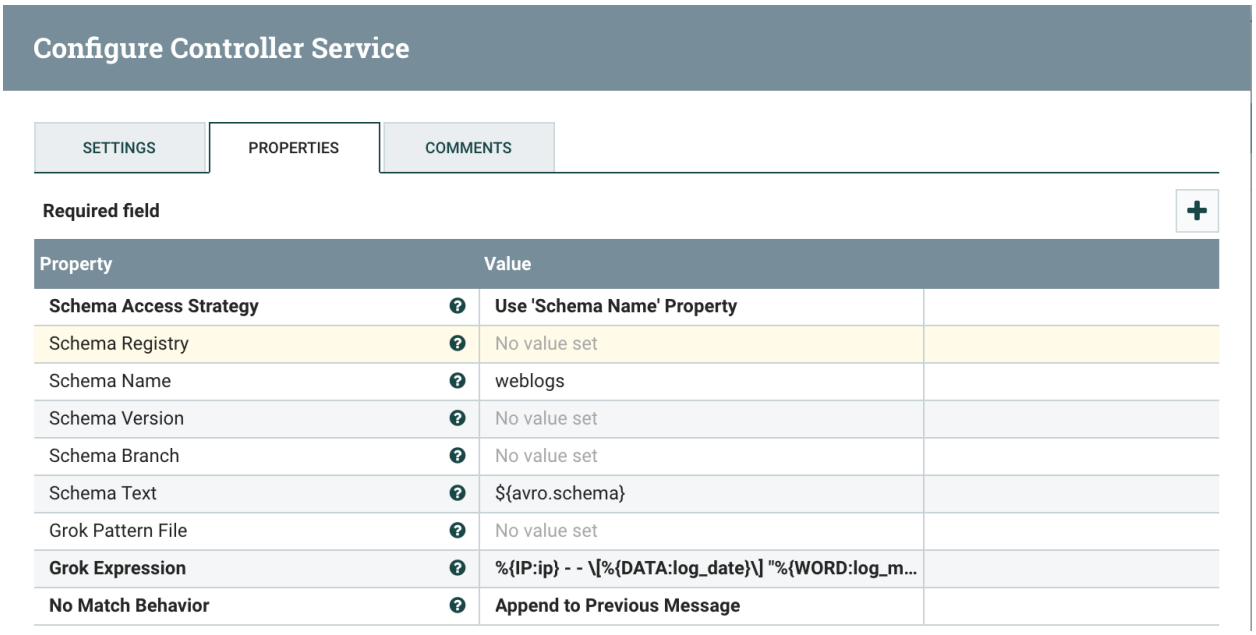
and set these properties:

Schema Access Strategy: **Use 'Schema Name' Property**

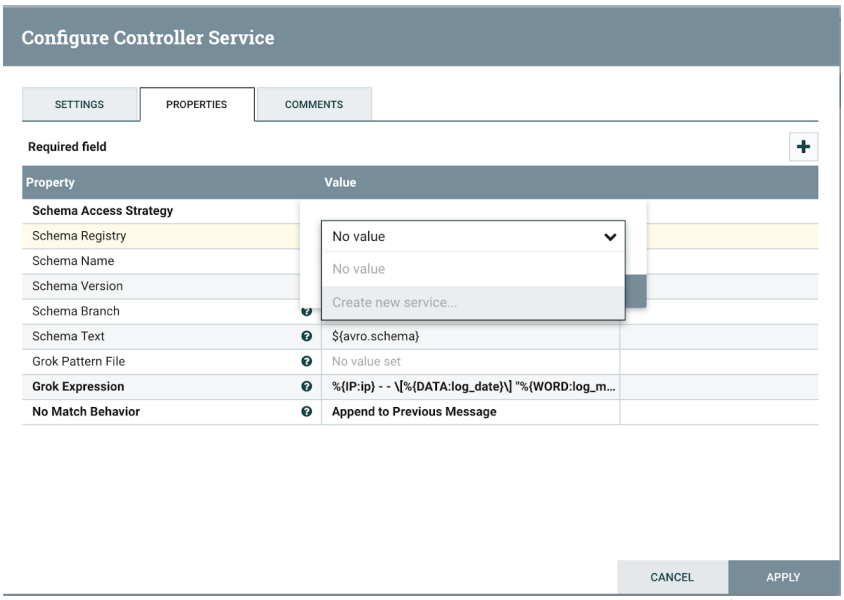
Schema Name: *weblogs*

Grok Expression:

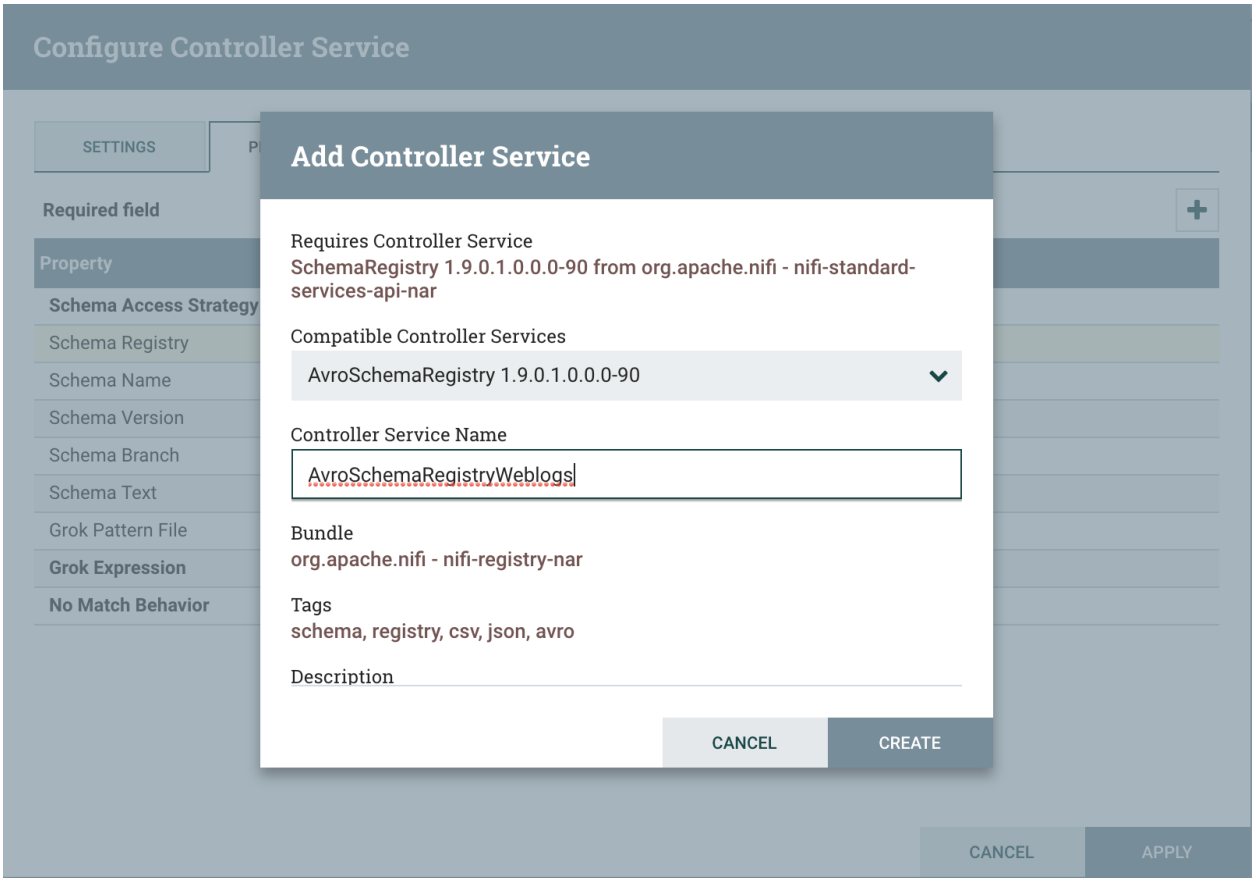
```
%{IP:ip} - - \[%{DATA:log_date}\] "%{WORD:log_method}
%{URIPATH:url} HTTP/%{NUMBER:http_version}" %{NUMBER:code1}
%{NUMBER:code2} "-" "%{DATA:user_agent}"
```



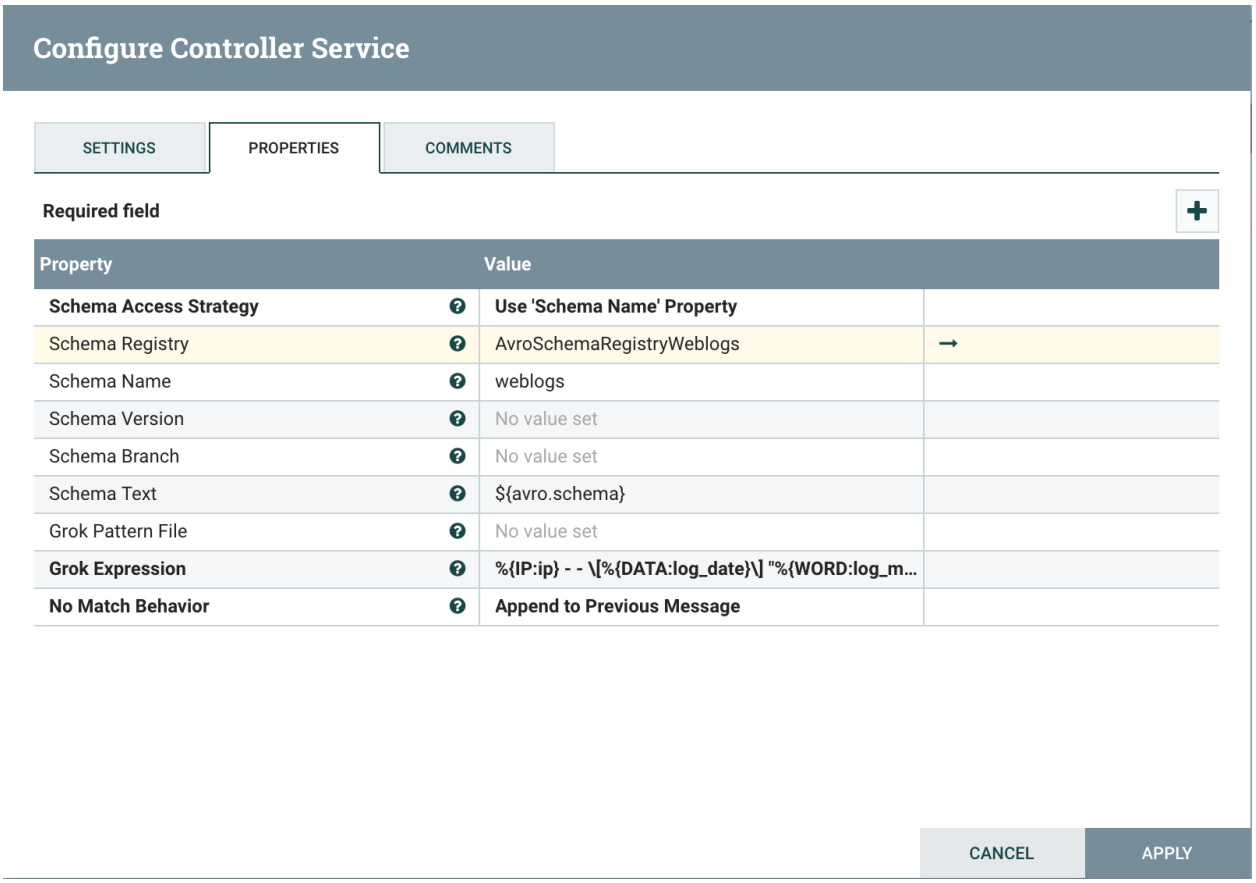
Also for the **Schema Registry** select **Create new service…**



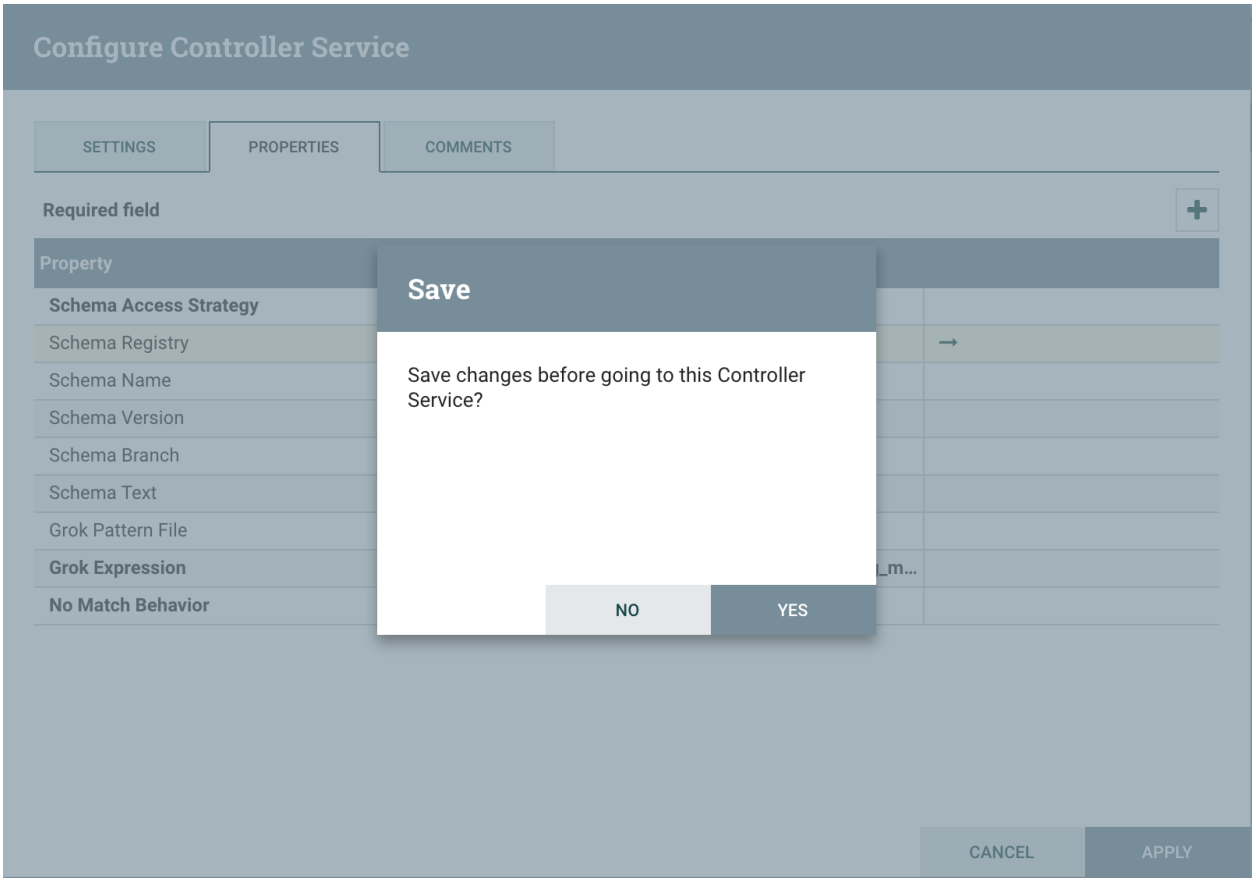
and select an **AvroSchemaRegistry**, give it a name (for example AvroSchemaRegistryWeblogs)



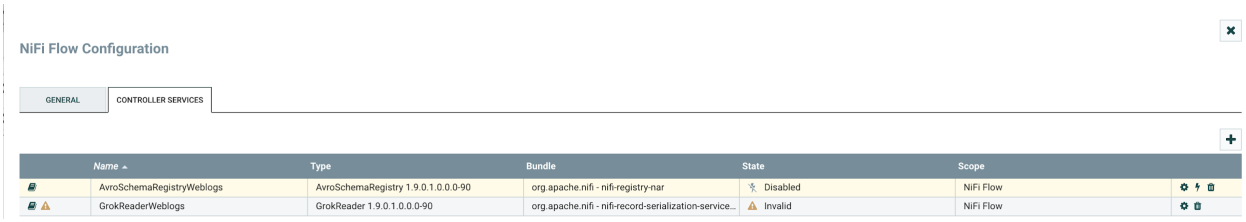
click **CREATE**



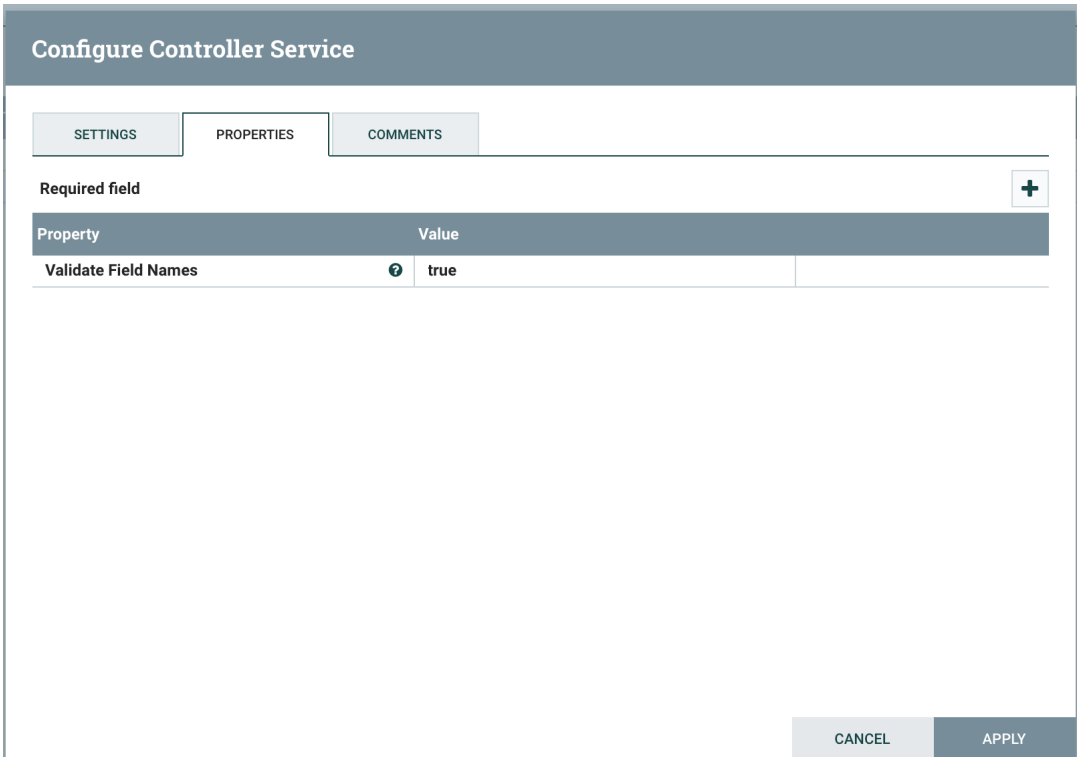
and configure as follows by clicking the arrow:



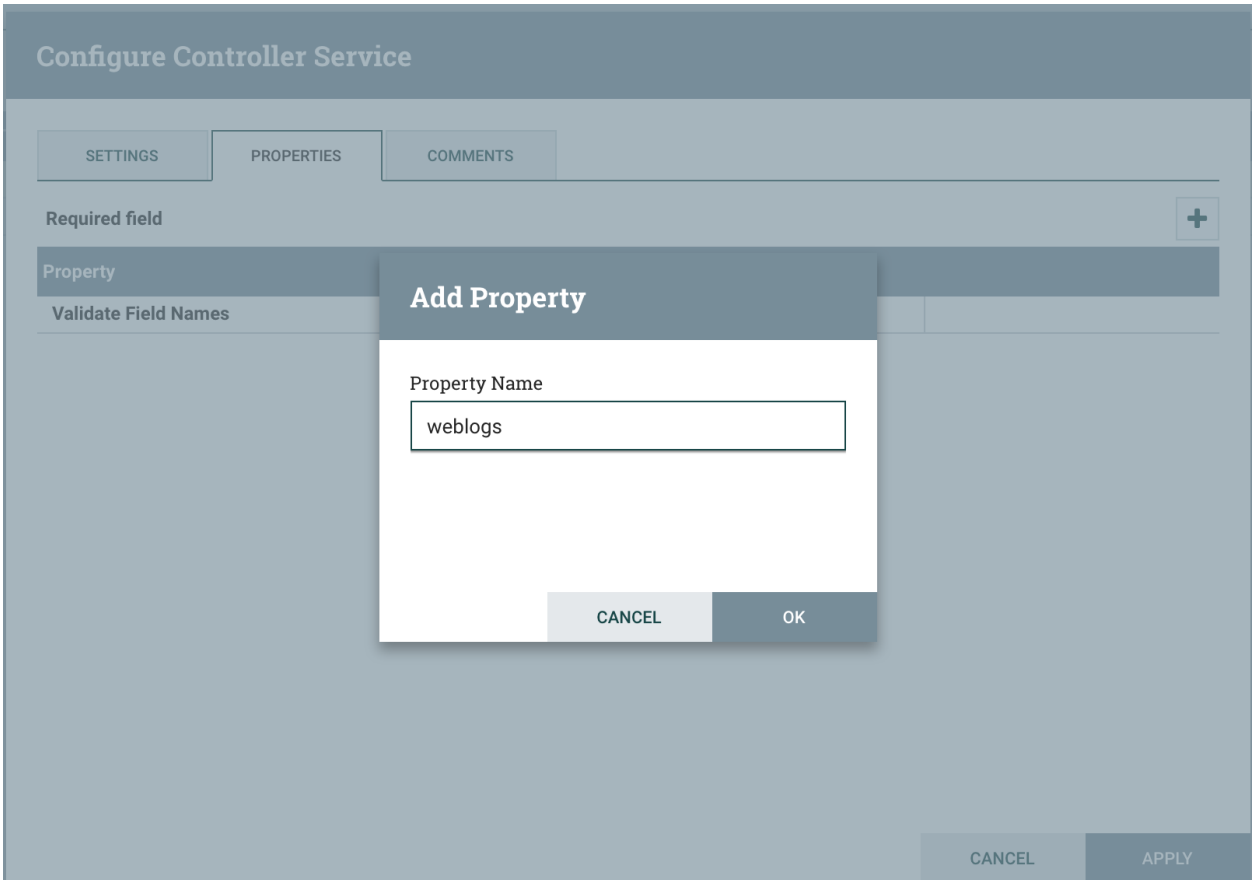
Save changes.



Click the **gear icon** for configuring the **AvroSchemaRegistryWeblogs**:



Click the **Plus (+) icon** to add a property *weblogs*

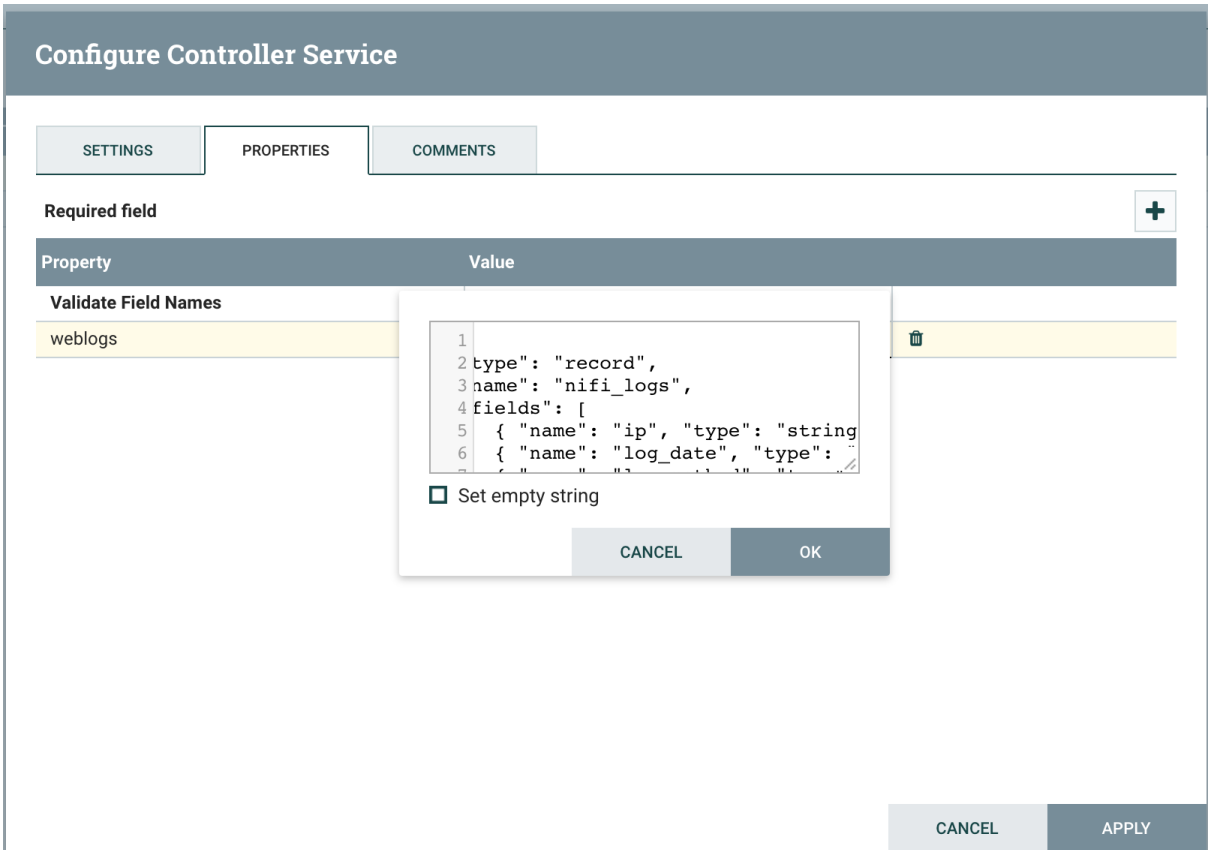


with value below that is the schema that describes our record.

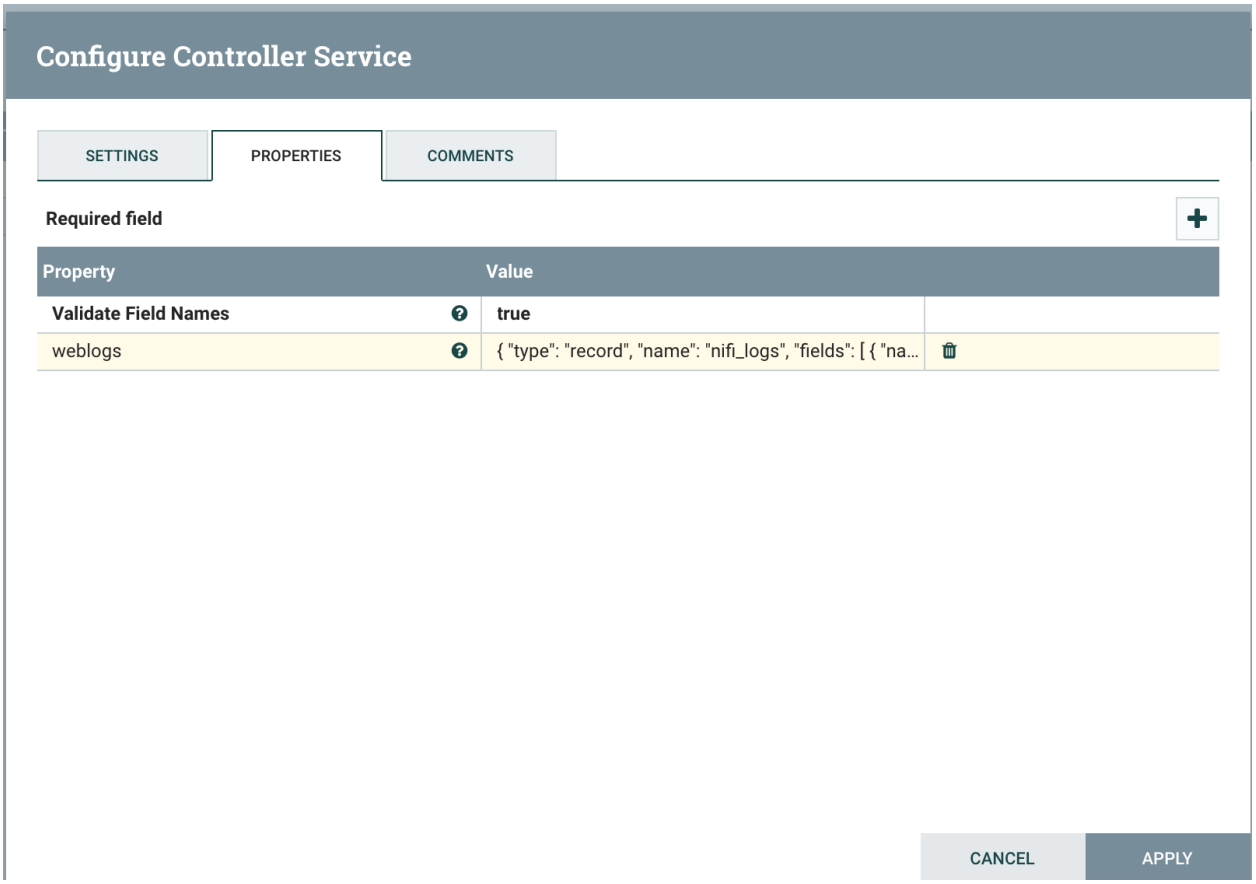
```
{
“type”: “record”,
“name”: “nifi_logs”,
“fields”: [
```

```
        {“name”: “ip”, ”type”: “string” },
        {“name”: “log_date”,” type”: “string” },
        {“name”: “log_method”, ”type”: “string” },
        {“name”: “url”, ”type”: “string” },
        {“name”: “http_version”, ”type”: “string” },
        {“name”: “code1”, ”type”: “string” },
        {“name”: “code2”, ”type”: “string” },
        {“name”: “user_agent”, ”type”: “string” },
```

```
        ]
}
```

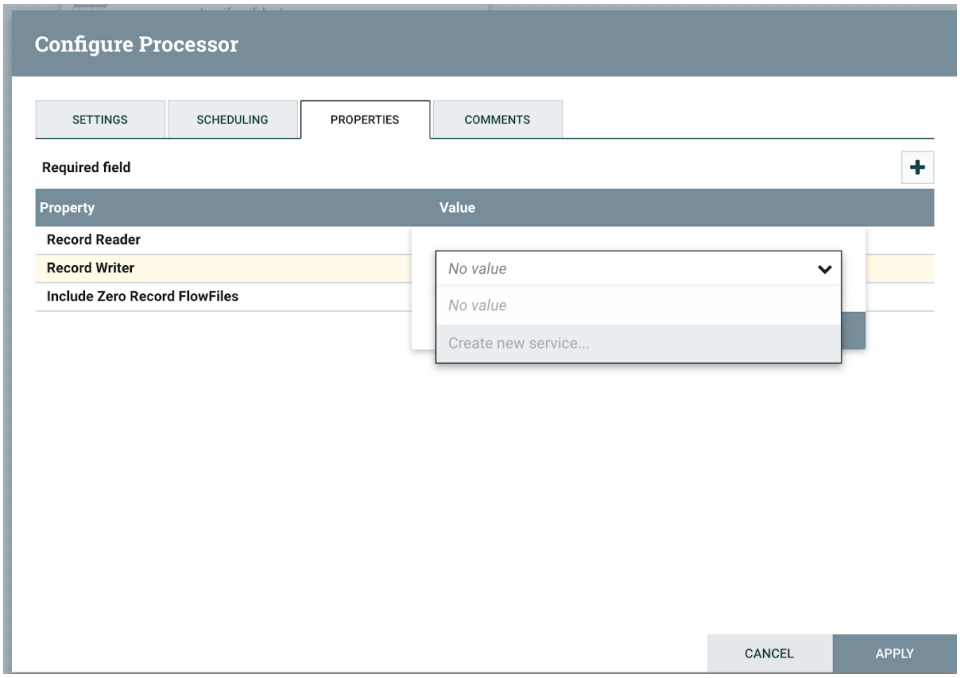

Configure Controller Service
SETTINGS
PROPERTIES
COMMENTS
Required field
Property
Value
Validate Field Names
weblogs
type": "record",
name": "nifi_logs",
fields": [
{ "name": "ip", "type": "string
{ "name": "log_date", "type":
Set empty string
CANCEL
OK
CANCEL
APPLY

click **OK**

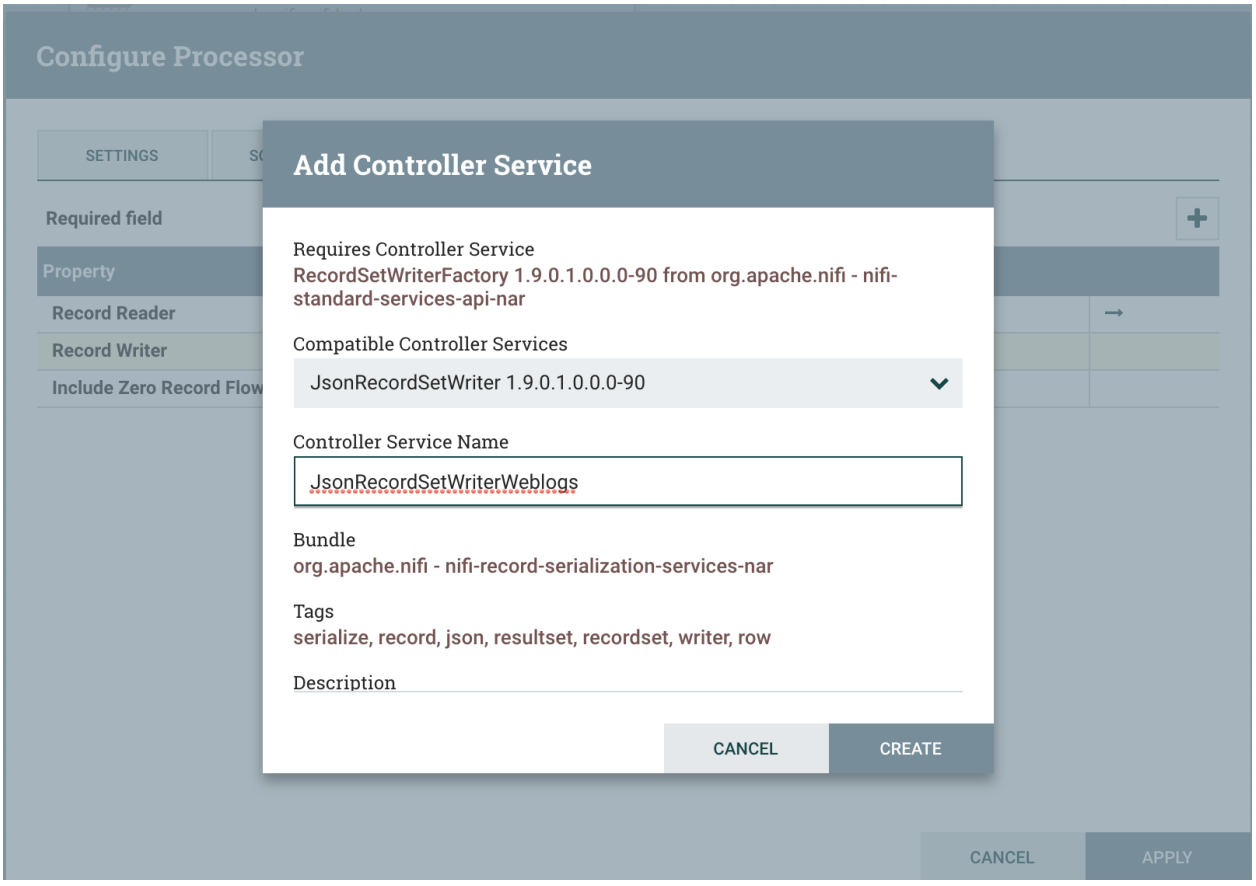


Then click **APPLY** and click on the **X** icon

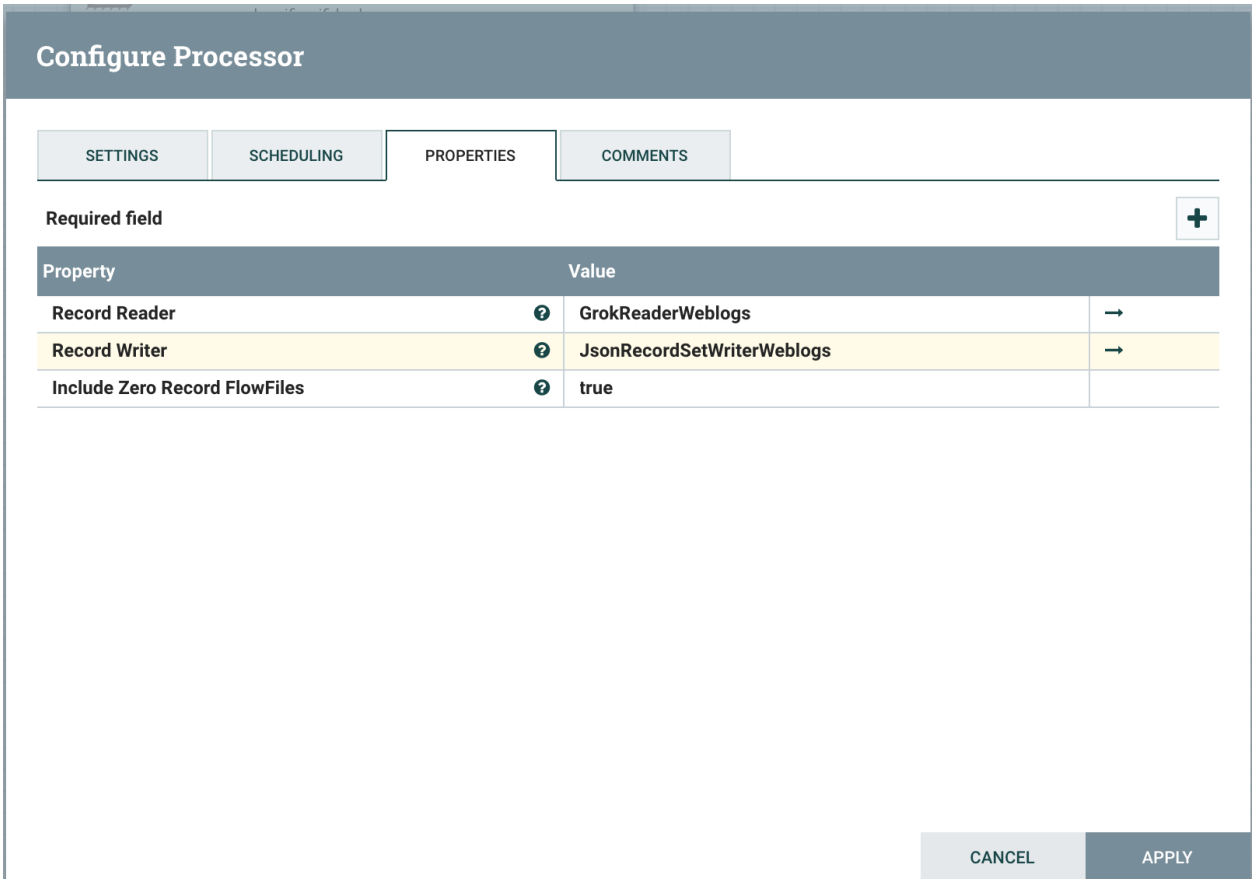
Then go back to the **ConvertRecord** and for the **Record Writer** create a new service



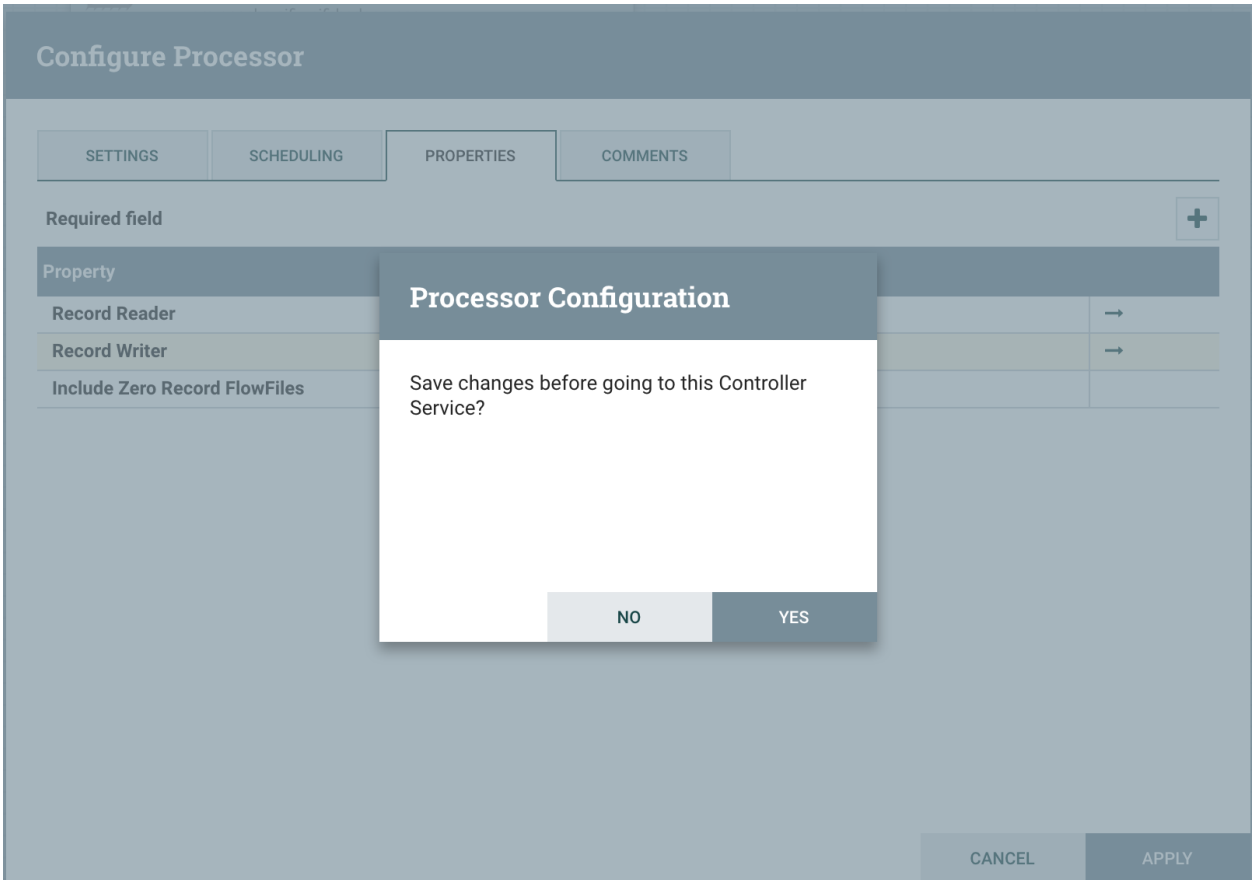
of type **JsonRecordSetWriter** (for example JsonRecordSetWriterWeblogs):



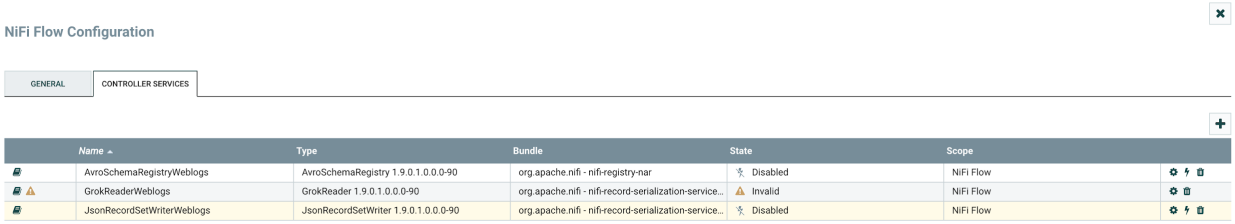
click **CREATE**



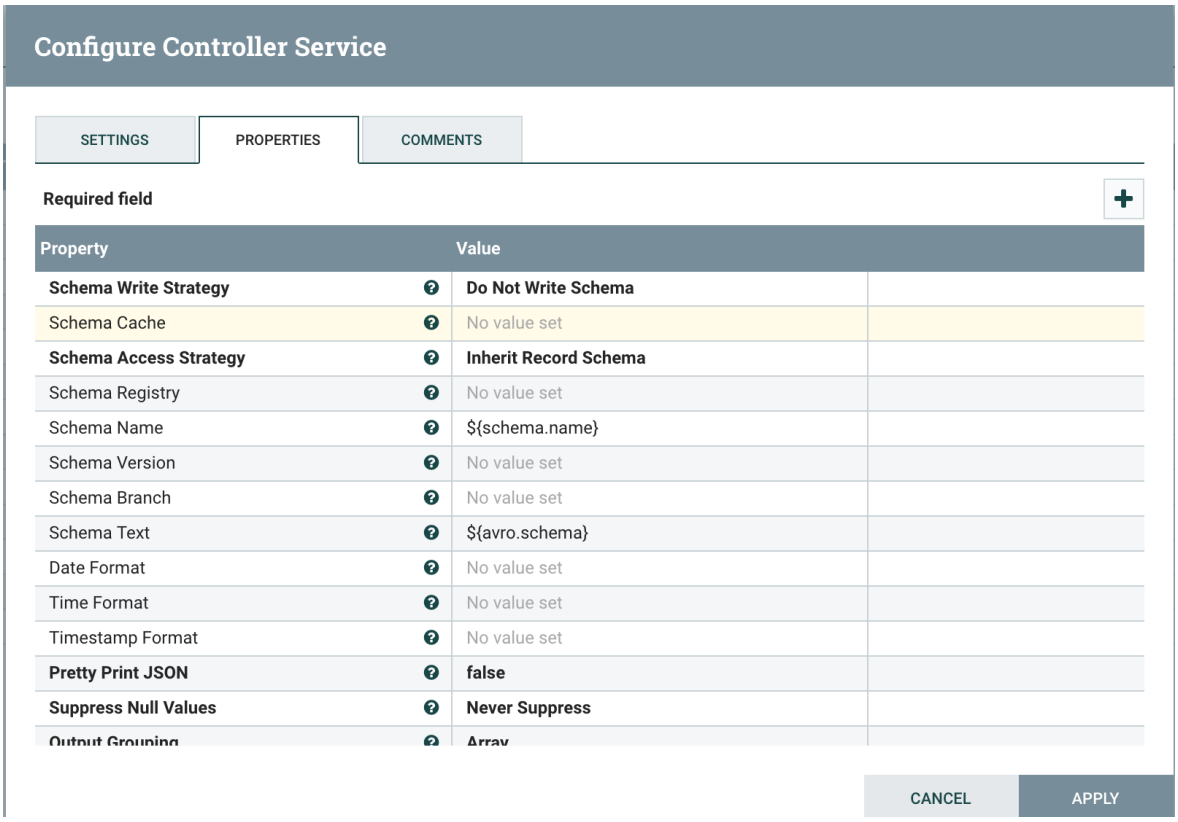
then configure by clicking the arrow and remember to save the changes:



Click **YES**

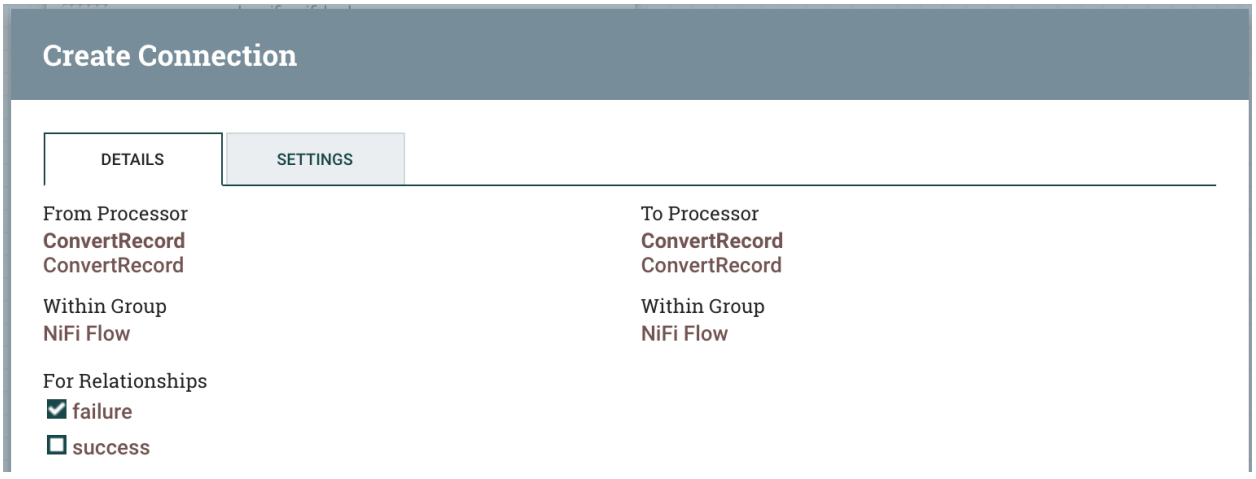


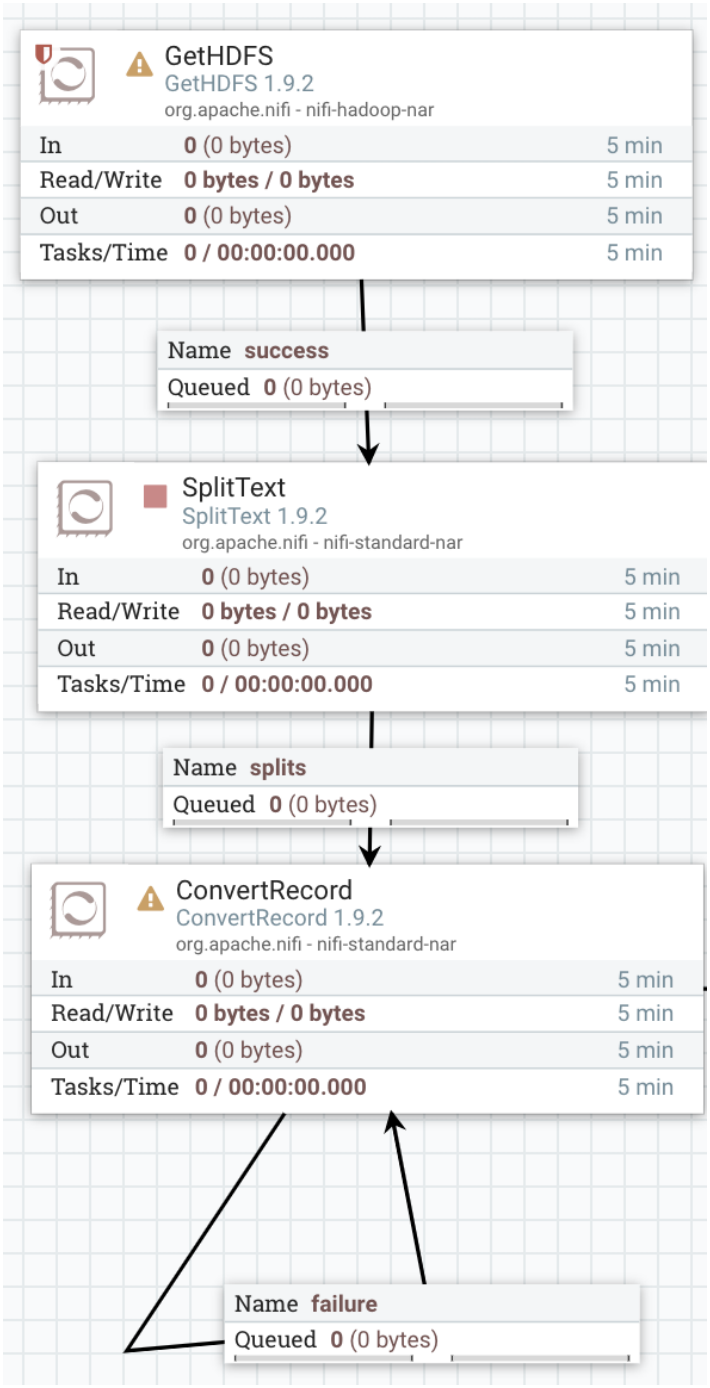
click the gear icon for the JsonRecordSetWriterWeblogs



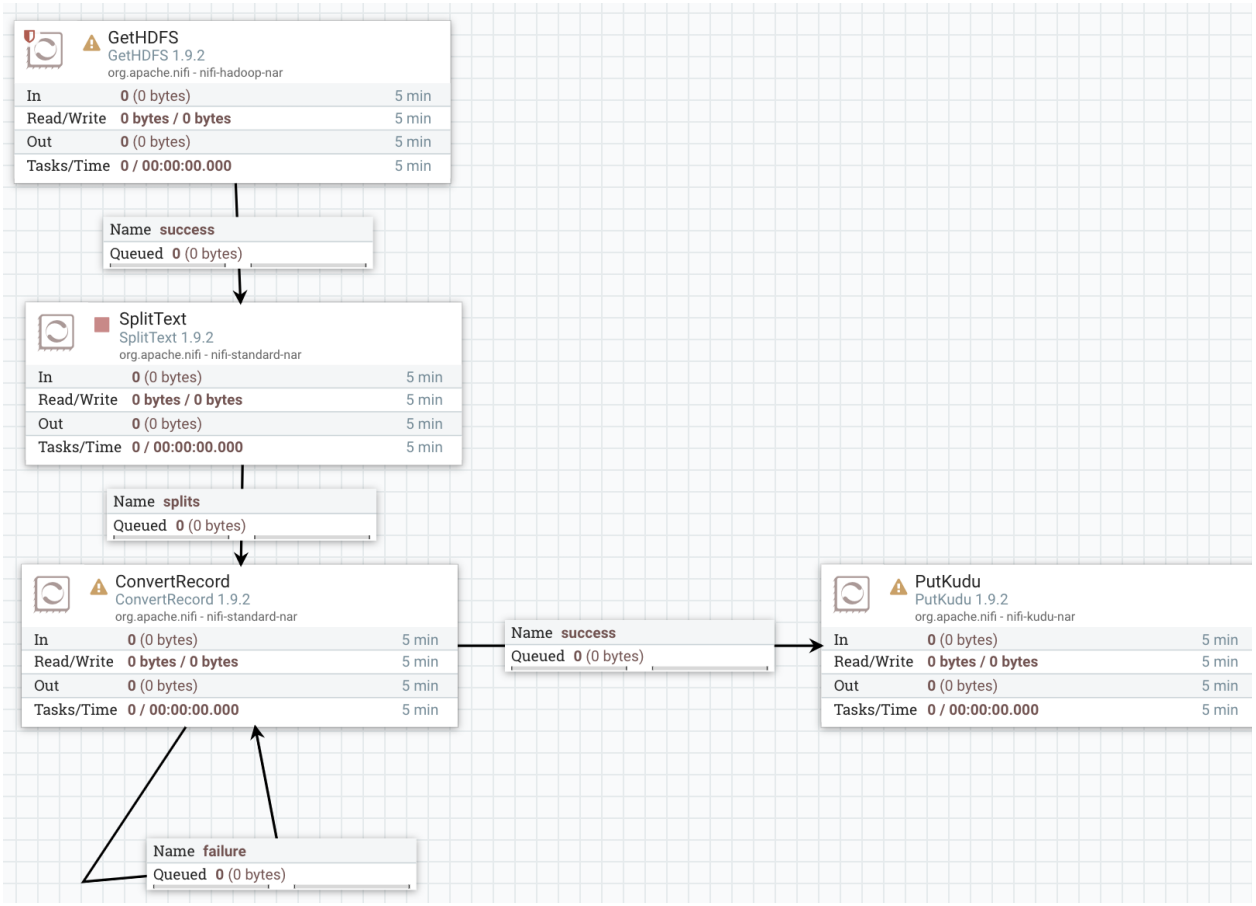
accept the default and click **APPLY**

Go back to the **ConvertRecord** and drag & drop a line from the **ConvertRecord** back to itself and check *failure*:

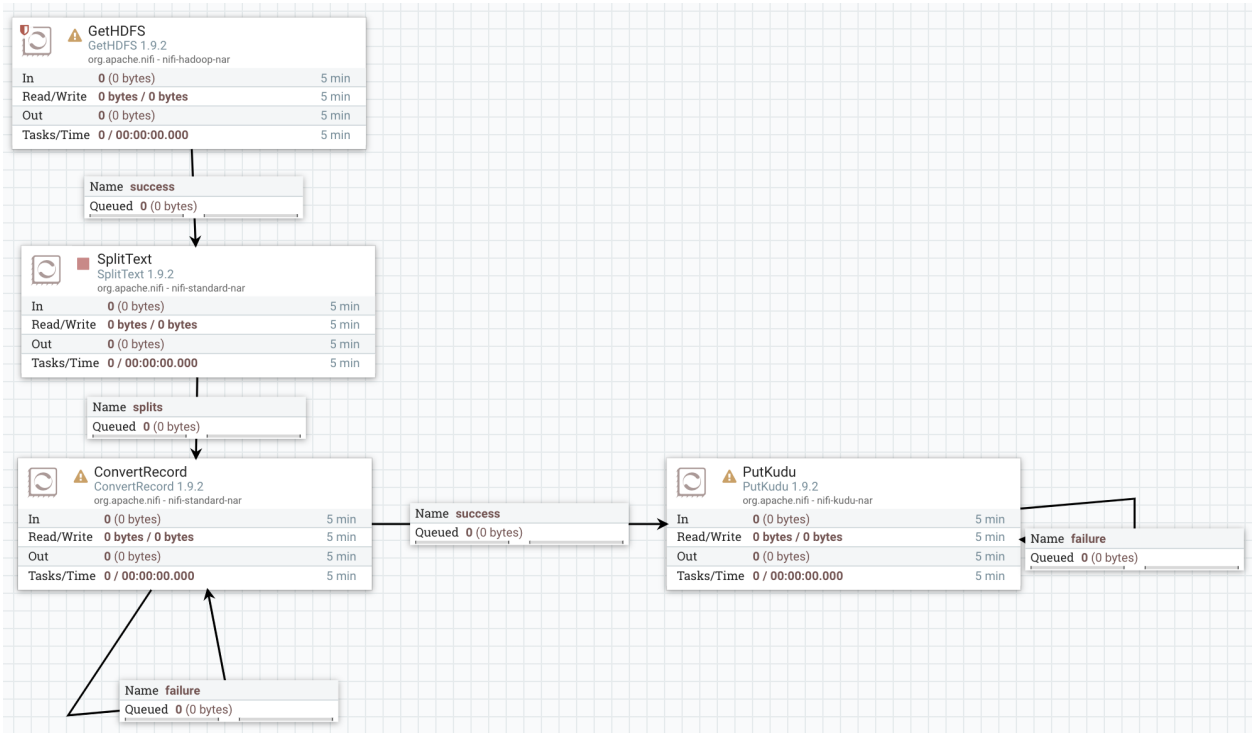




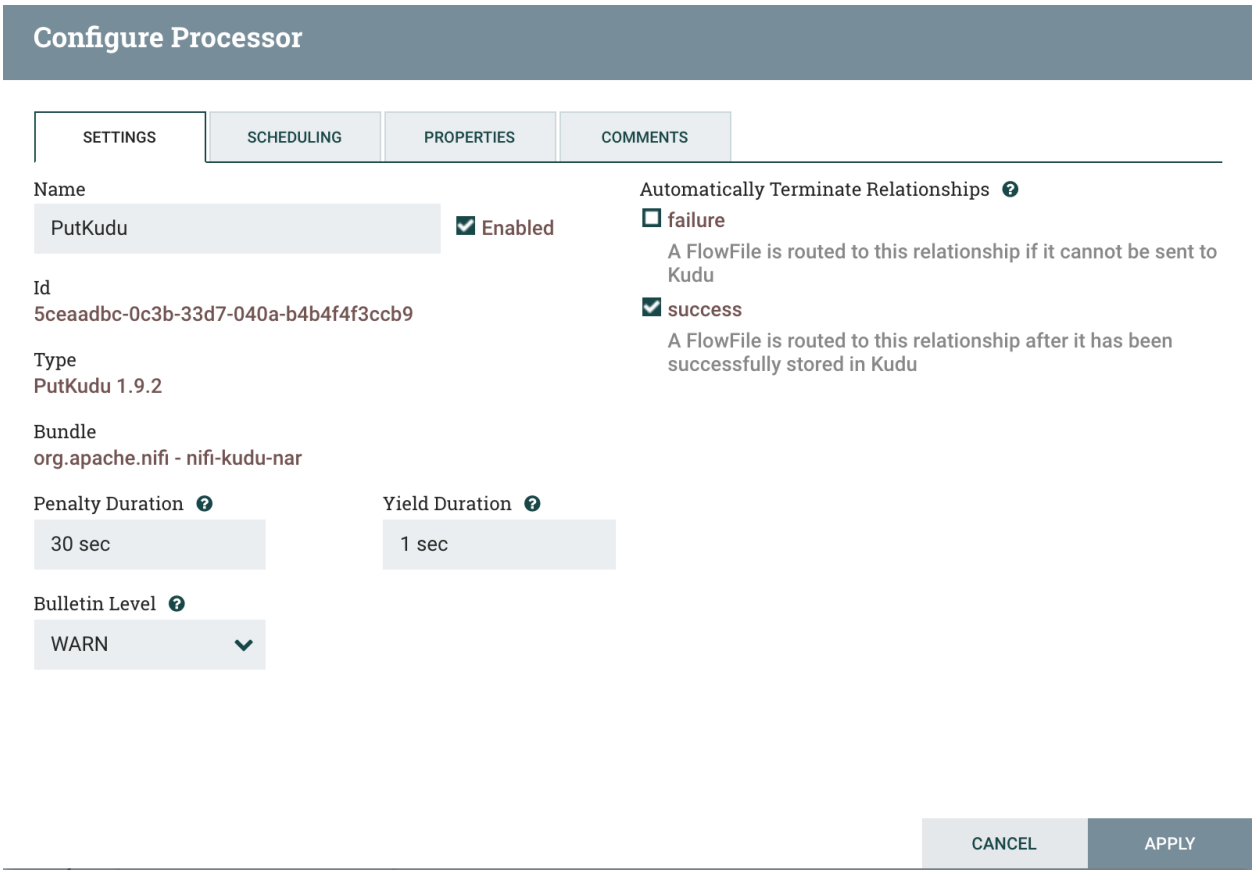
Then add the **PutKudu** processor and link it to the **ConvertRecord**:



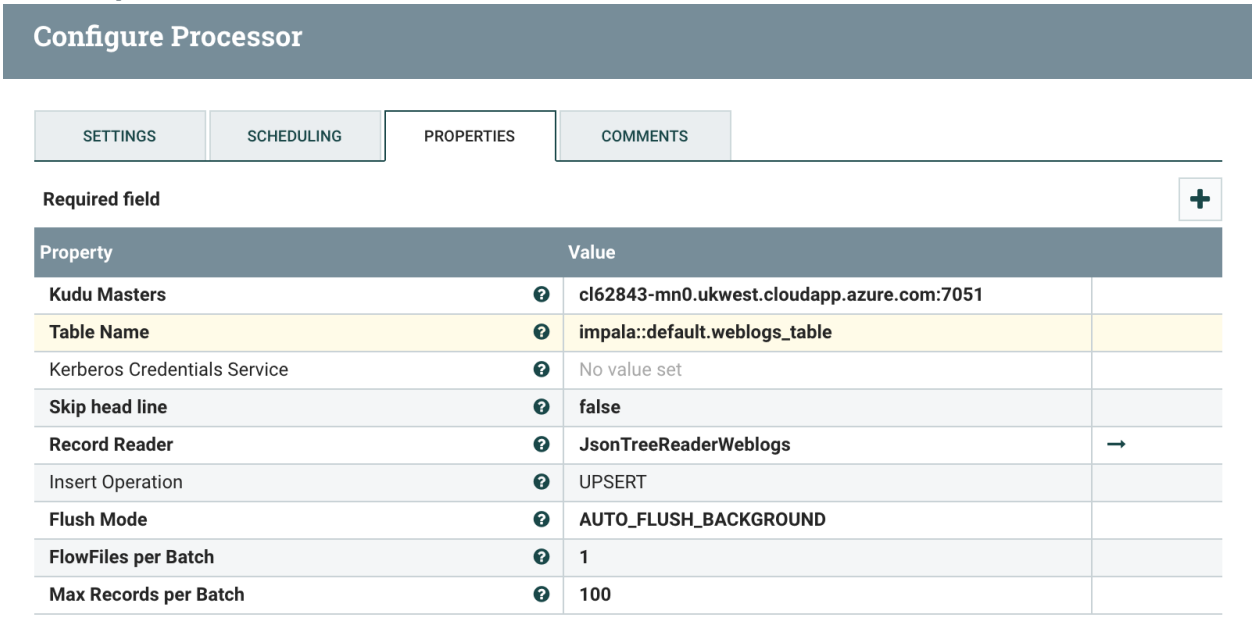
and also for **PutKudu** drag & drop a connection back to itself and check **failure**:



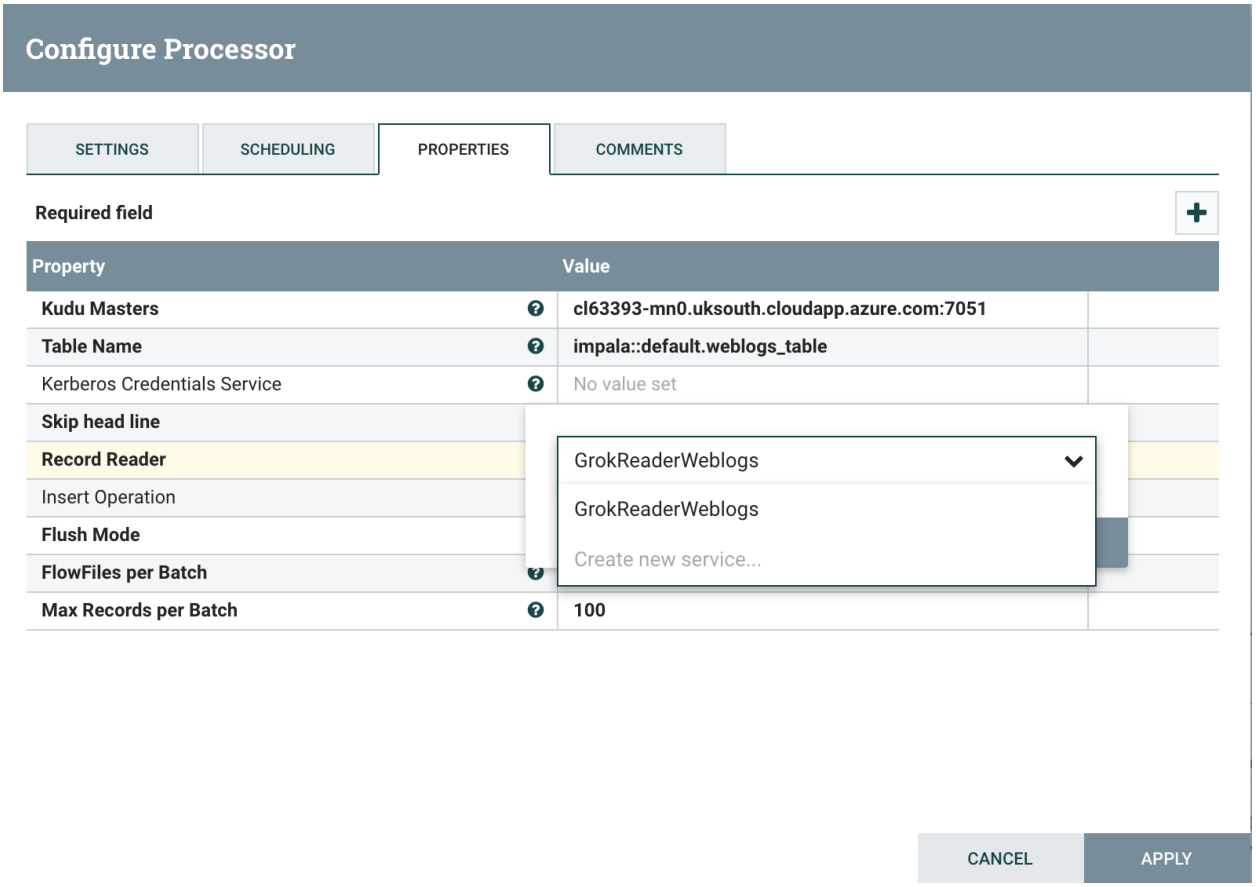
Double click on **PutKudu** and go to the **SETTINGS** tab and automatically terminate in case of success:



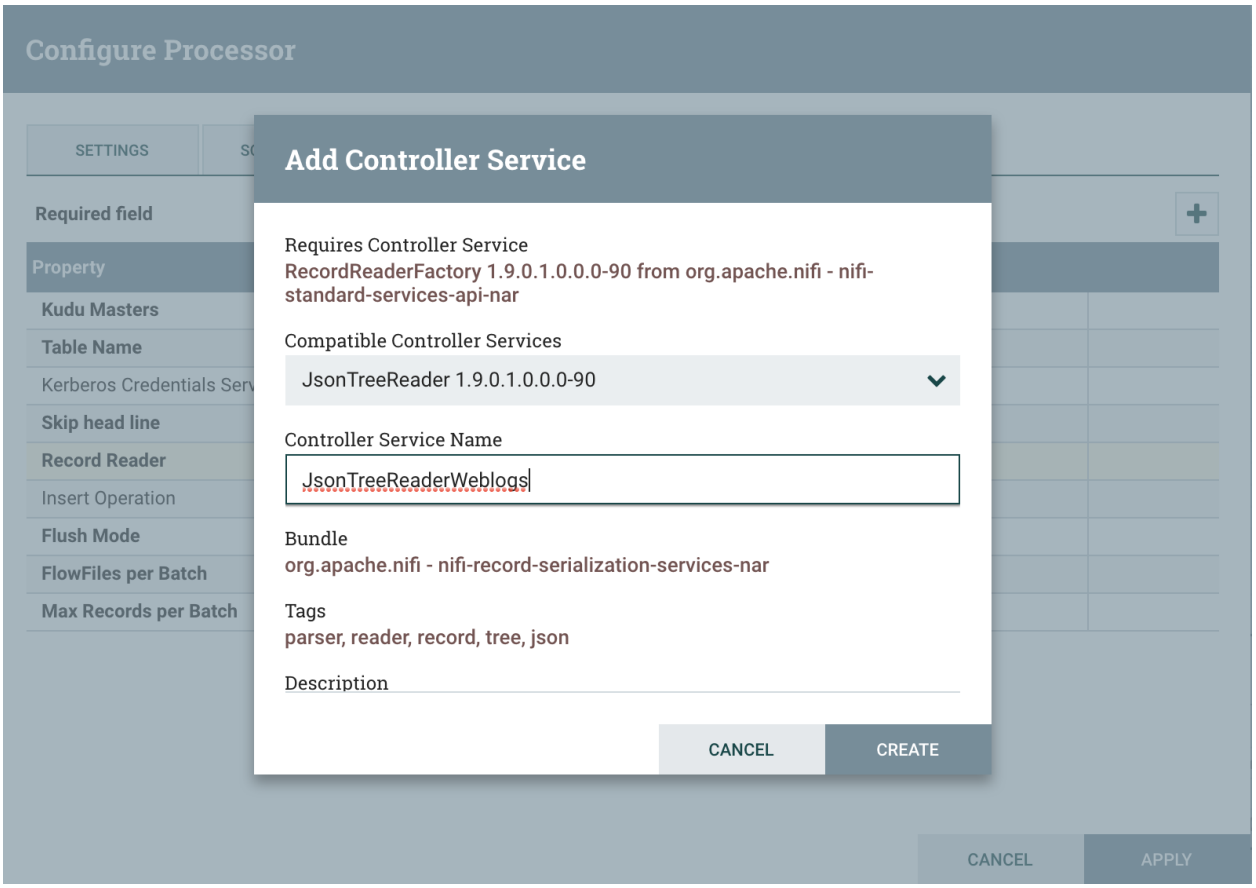
Then go to the **PROPERTIES** tab and the these properties:
Kudu Masters: *<address of your Master node>:7051* (in my case flamb2-mn0.westeurope.cloudapp.azure.com:7051)
Table Name: impala::default.weblogs_table
Insert Operation: UPSERT



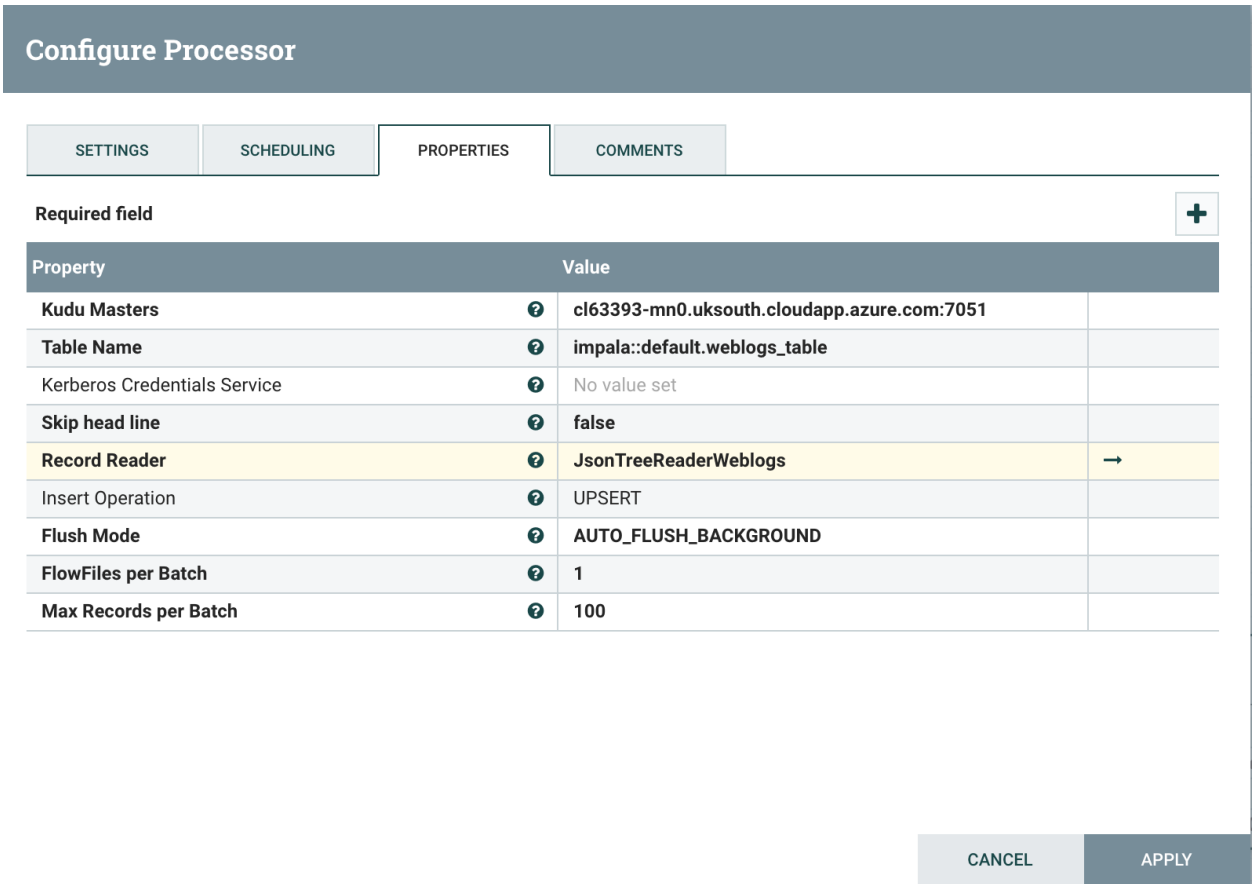
For the **Record Reader** click on **Create new services…**



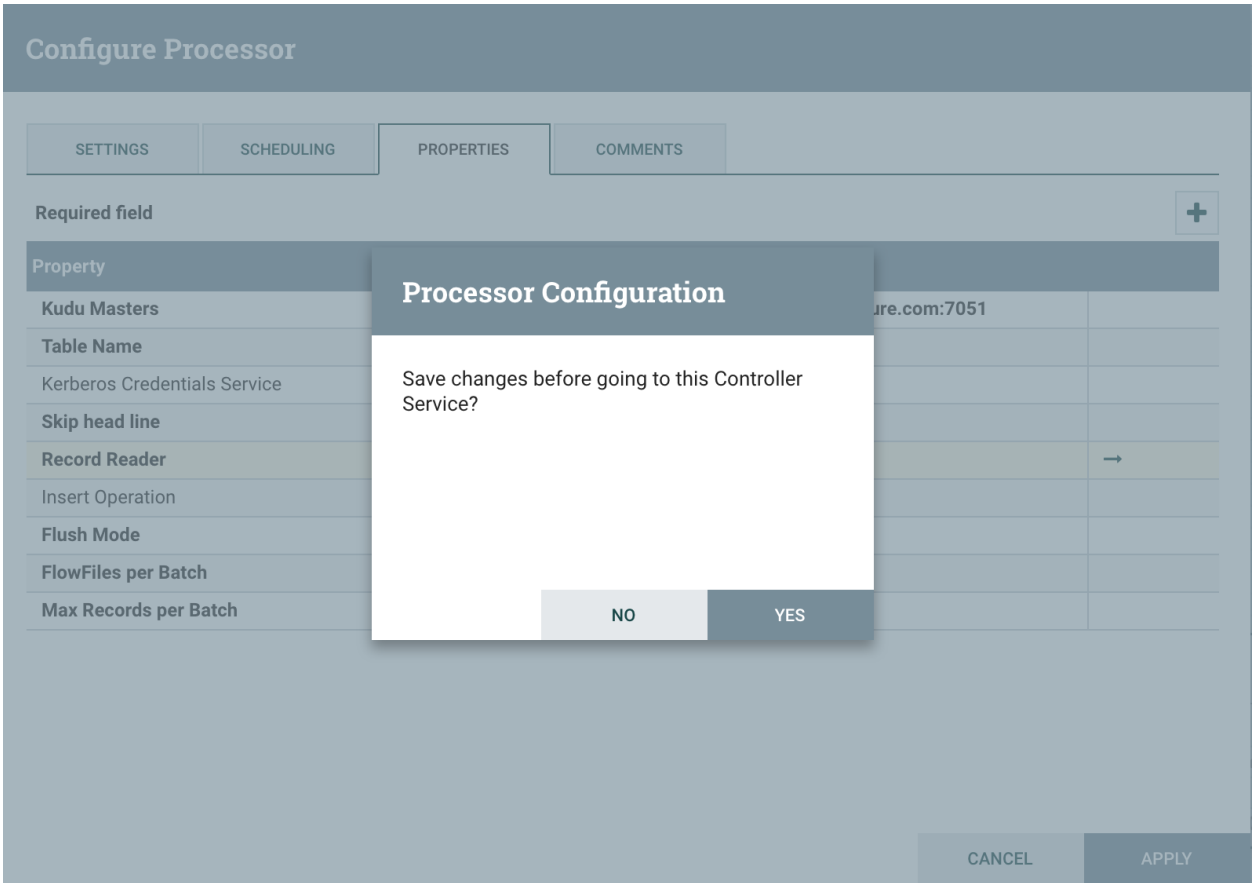
In the **Compatible Controller Services** select **JsonTreeReader** and give it a name (JsonTreeReaderWeblogs)

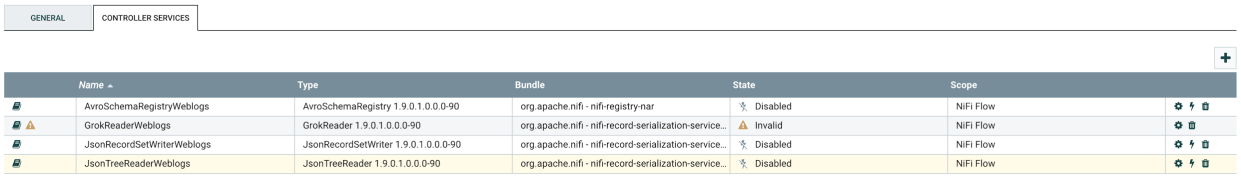

Configure Processor
SETTINGS
Required field
Property
Kudu Masters
Table Name
Kerberos Credentials Serv
Skip head line
Record Reader
Insert Operation
Flush Mode
FlowFiles per Batch
Max Records per Batch
Add Controller Service
Requires Controller Service
RecordReaderFactory 1.9.0.1.0.0.0-90 from org.apache.nifi - nifi-standard-services-api-nar
Compatible Controller Services
JsonTreeReader 1.9.0.1.0.0.0-90
Controller Service Name
JsonTreeReaderWeblogs
Bundle
org.apache.nifi - nifi-record-serialization-services-nar
Tags
parser, reader, record, tree, json
Description
CANCEL
CREATE
CANCEL
APPLY

CREATE

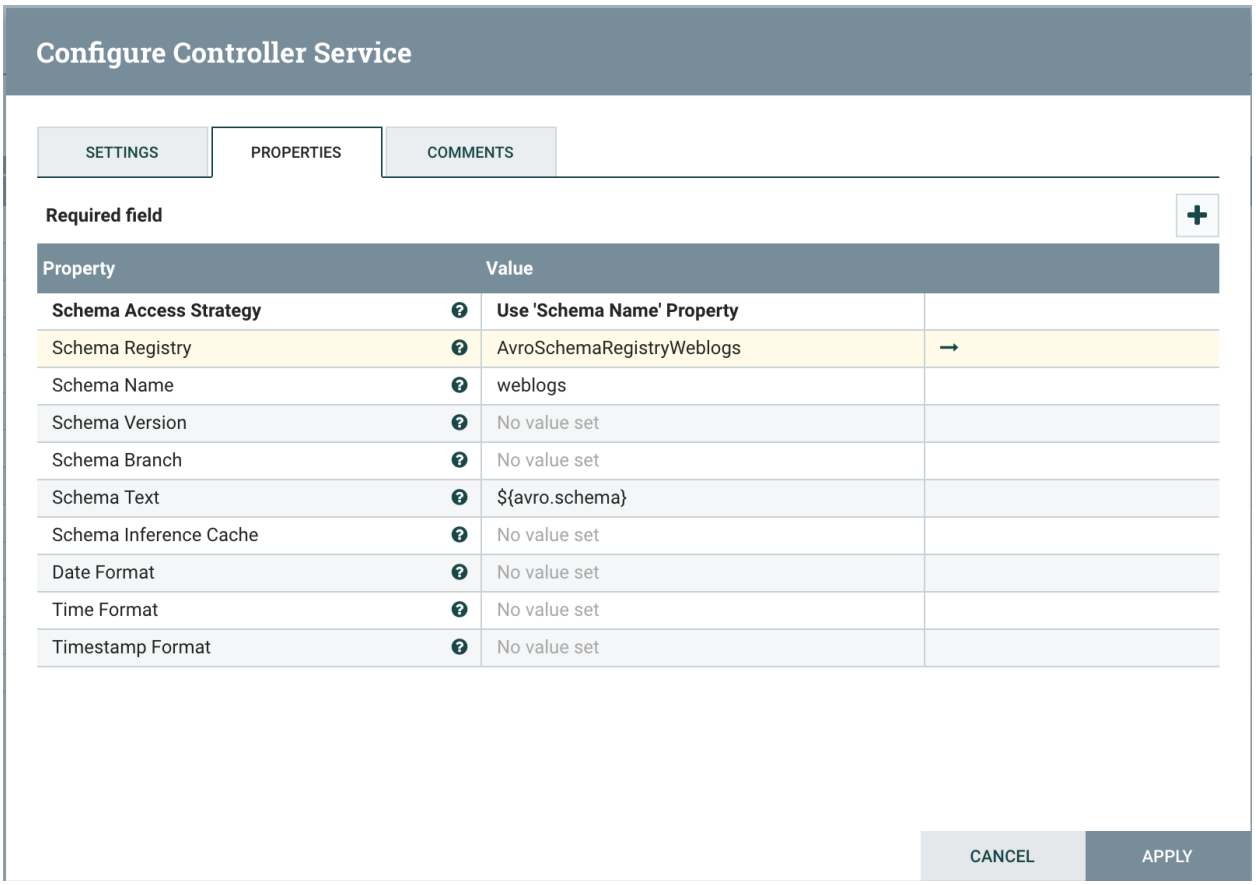


arrow and save



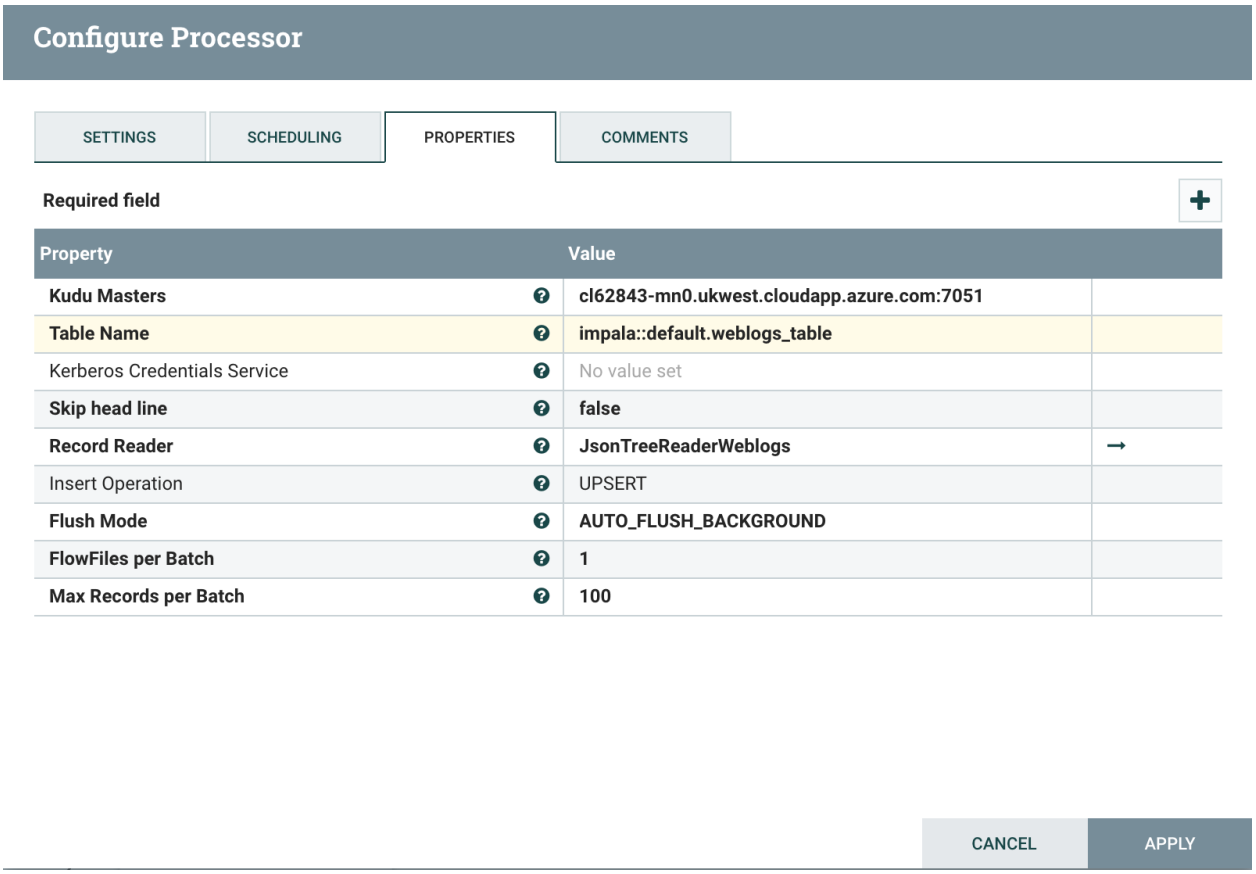


then configure it as follows

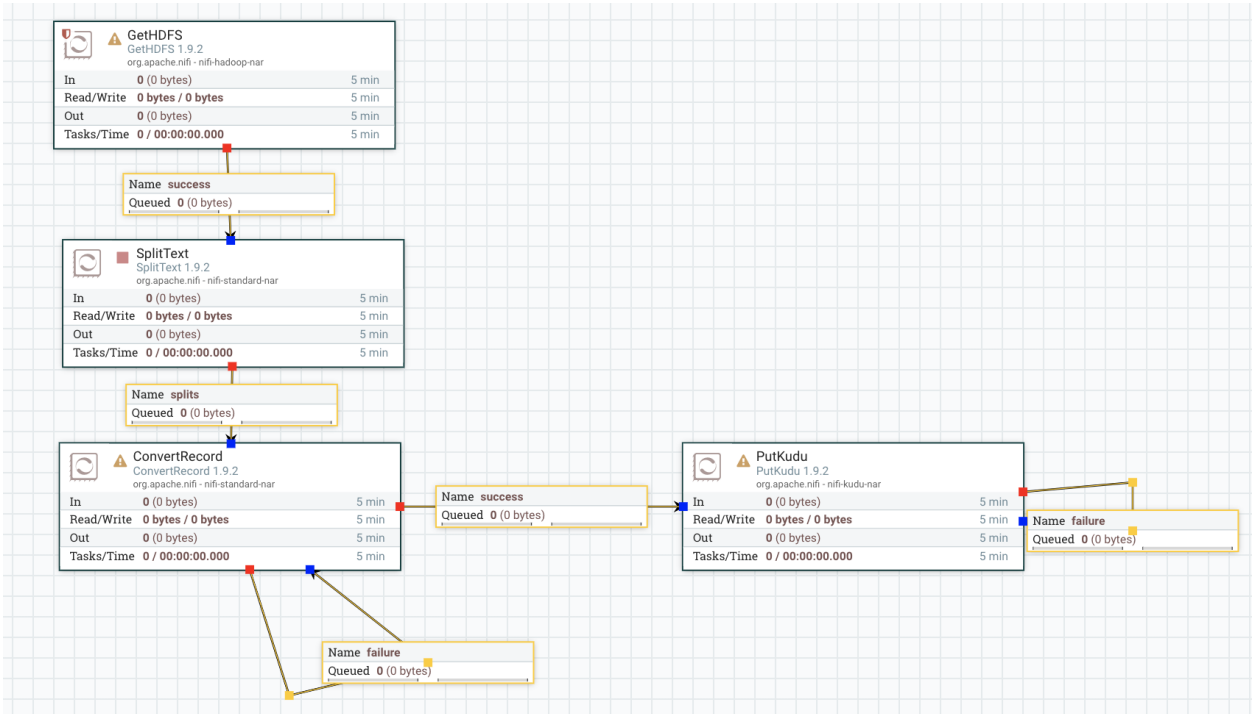


where
Schema Registry is **AvroSchemaRegistryWeblogs** (the one also used before)
Schema Name is *weblogs*

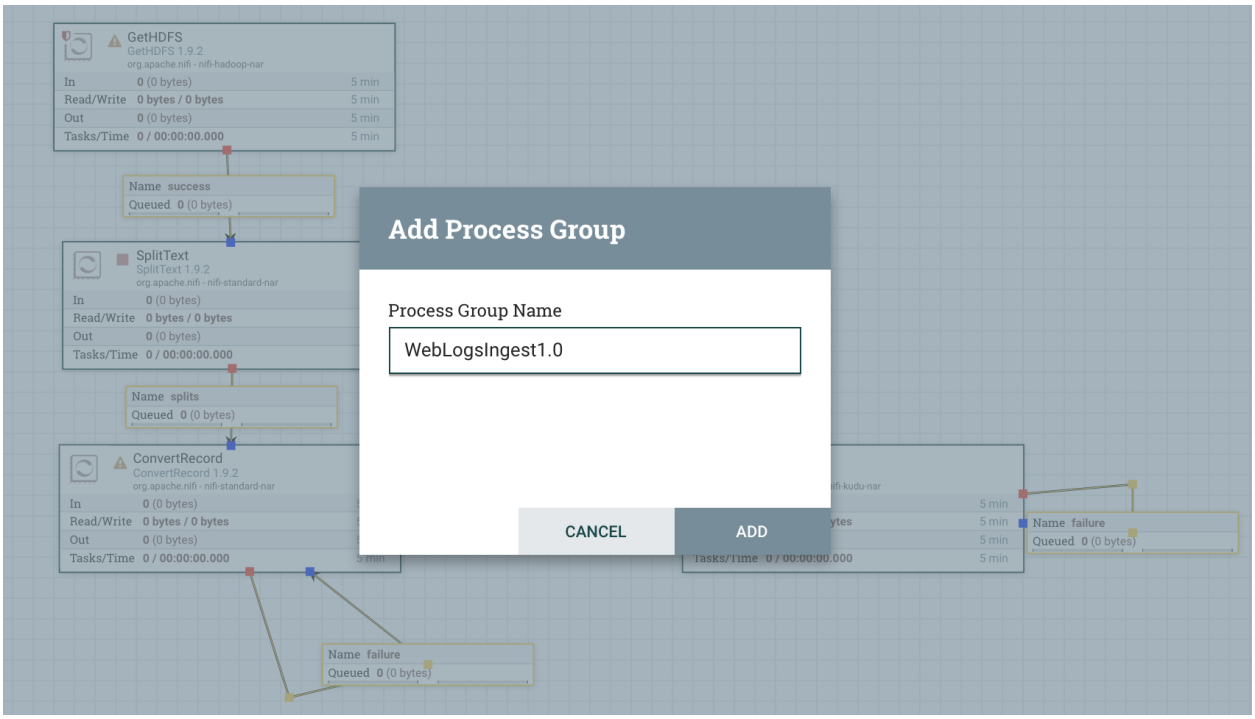
Click **APPLY a**nd go back to the **PutKudu** processor



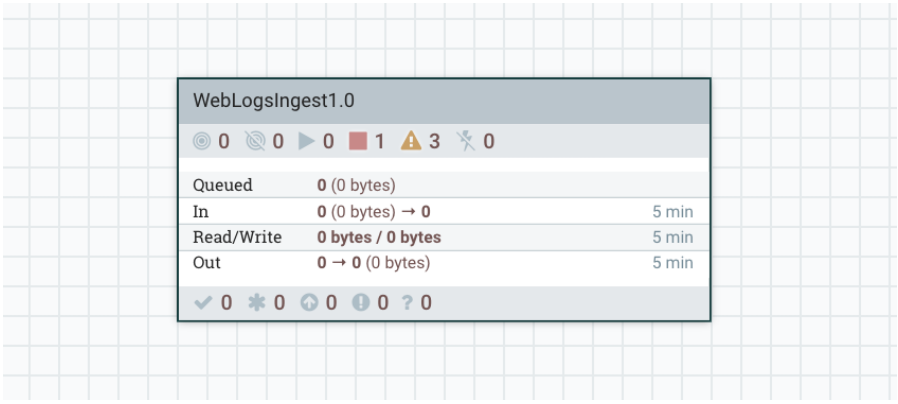
Now that the flow is finished, let's create a Group. With the mouse select a point in the canvas, press **SHIFT** and hold down the mouse and select your flow:



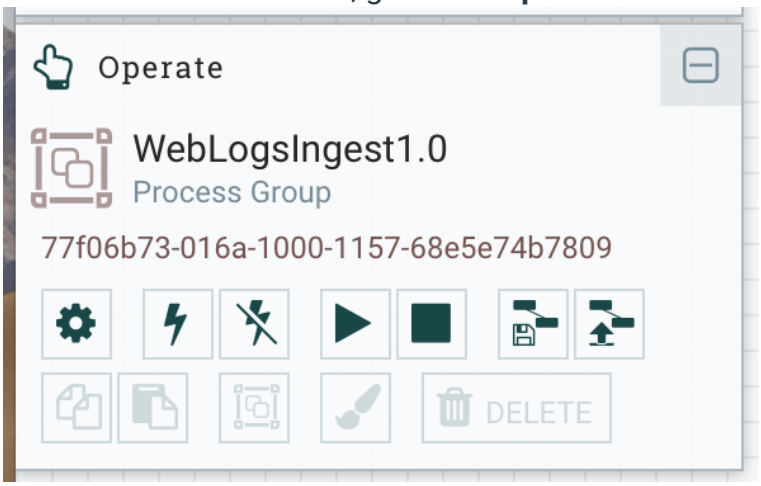
Then right click and click **Group** and give it a name:



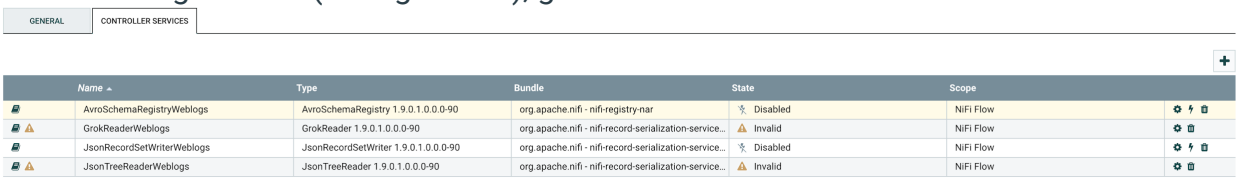
click **ADD**



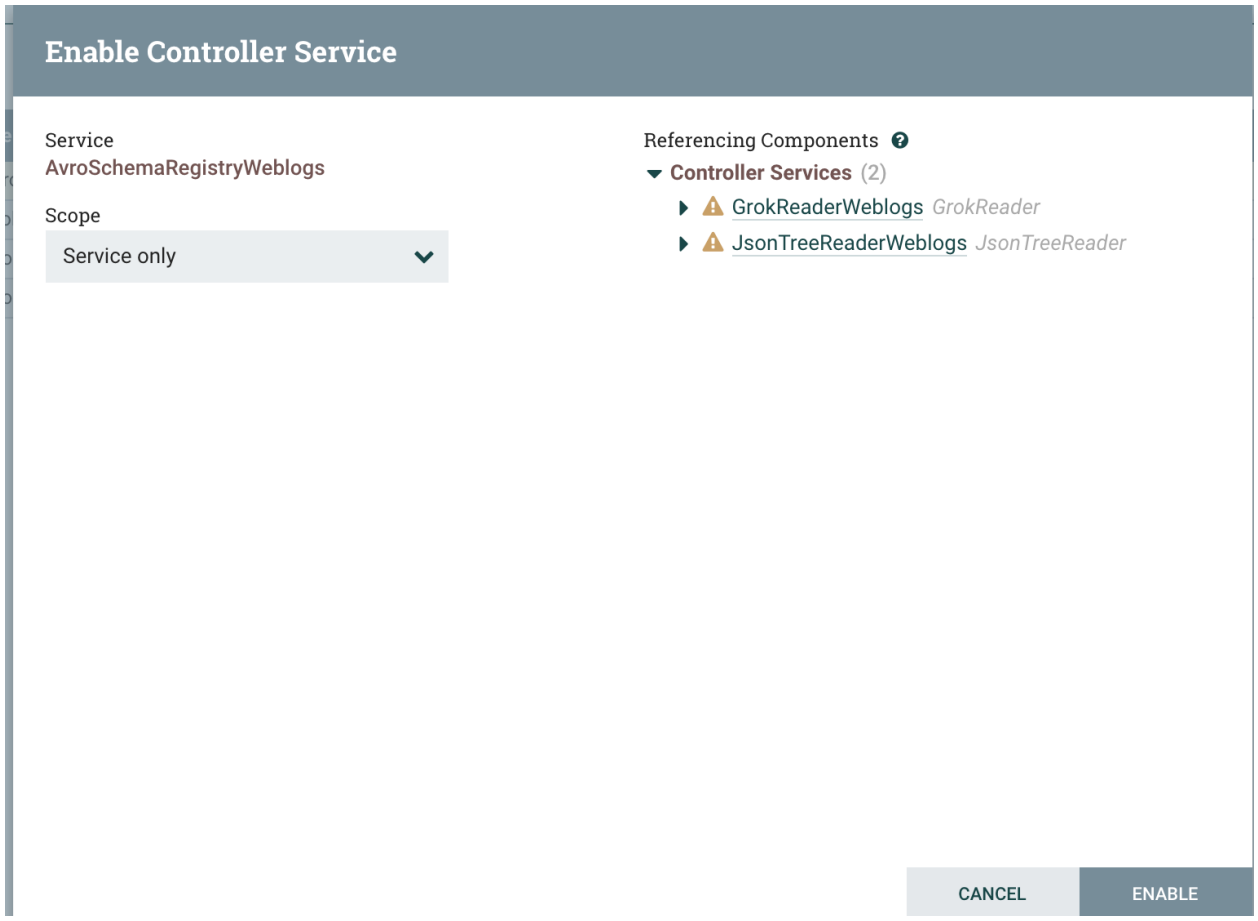
Now we can run the flow, go to the **Operate** box:



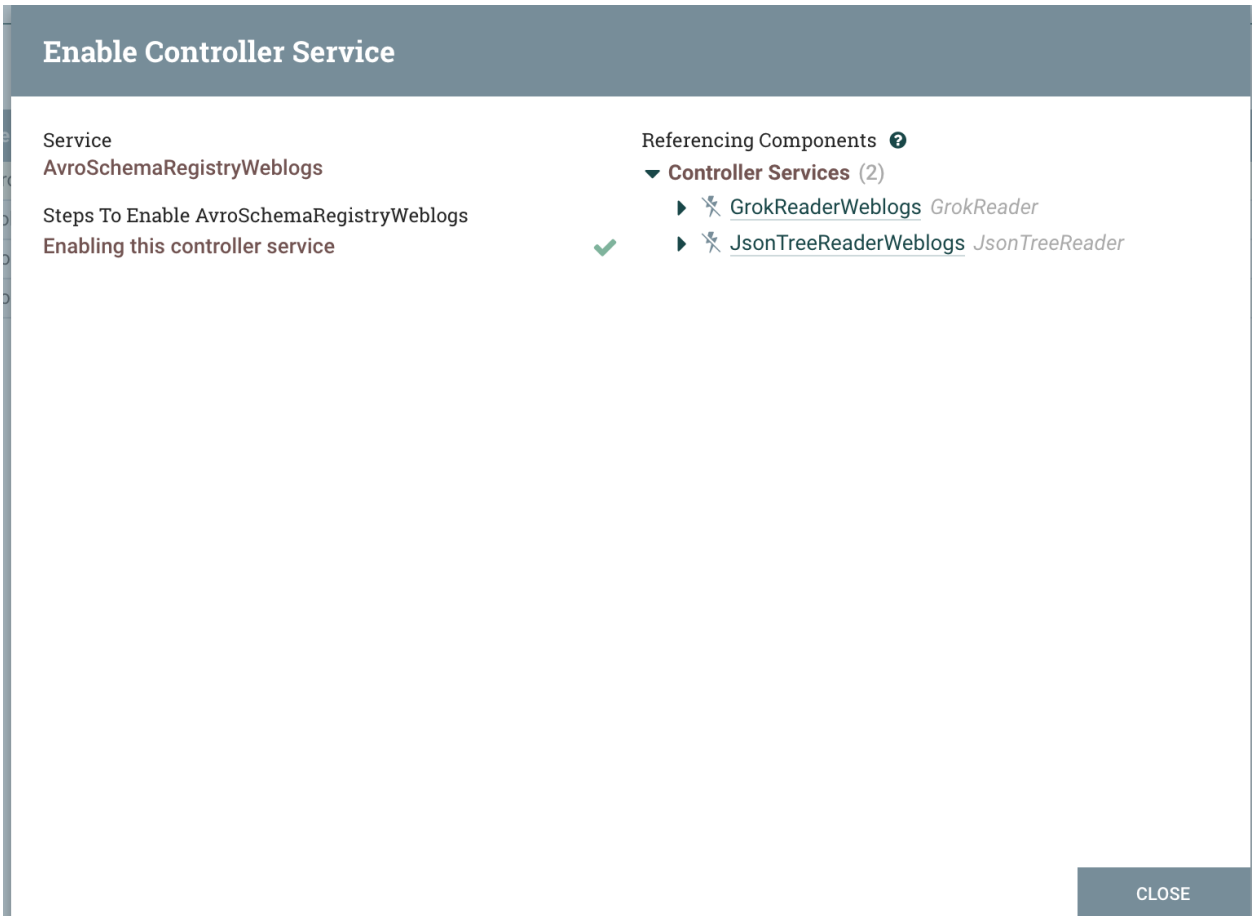
Click on the gear icon (configuration), go to the **CONTROLLER SERVICES** tab



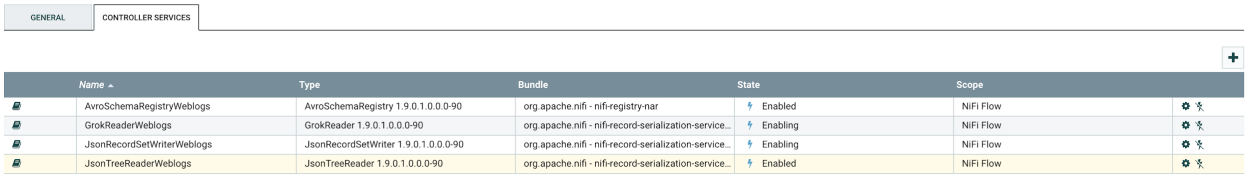
and Enable those that we have created by clicking the lightning icon.


Enable Controller Service
Service
AvroSchemaRegistryWeblogs
Scope
Service only
Referencing Components
Controller Services (2)
GrokReaderWeblogs GrokReader
JsonTreeReaderWeblogs JsonTreeReader
CANCEL
ENABLE

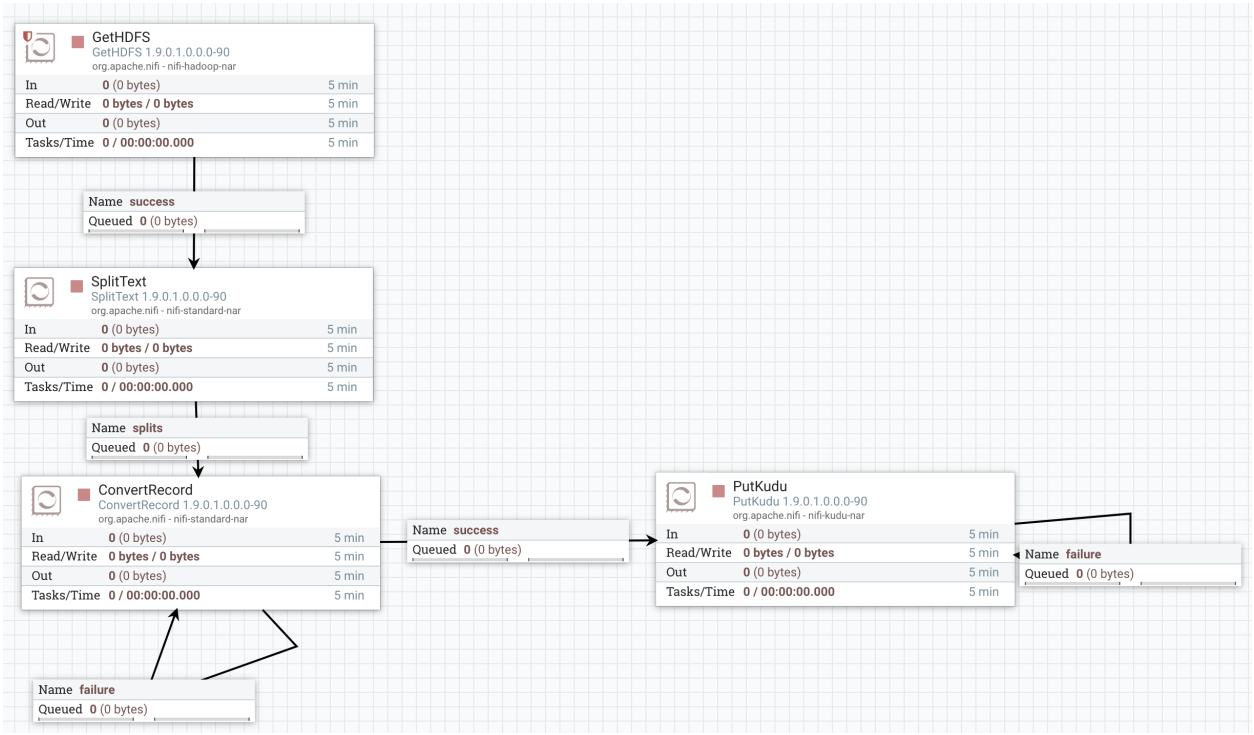
ENABLE



CLOSE and enable all the others



Go back to the flow and now you should see that all the processors are the red square icon and no more the warning icon

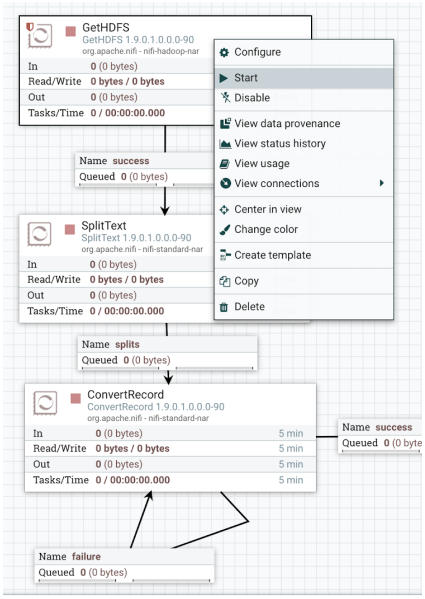


Now our flow is ready to run….

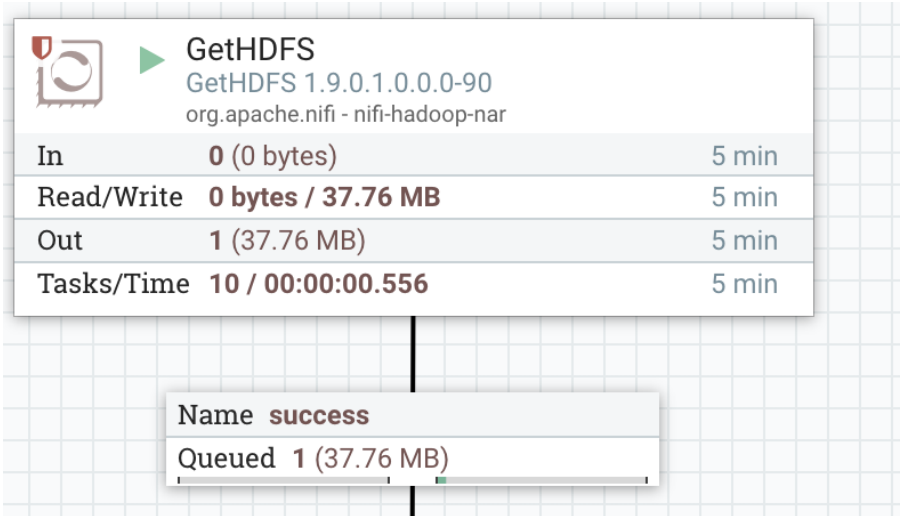
Run the Flow

We are going to run the flow step by step.

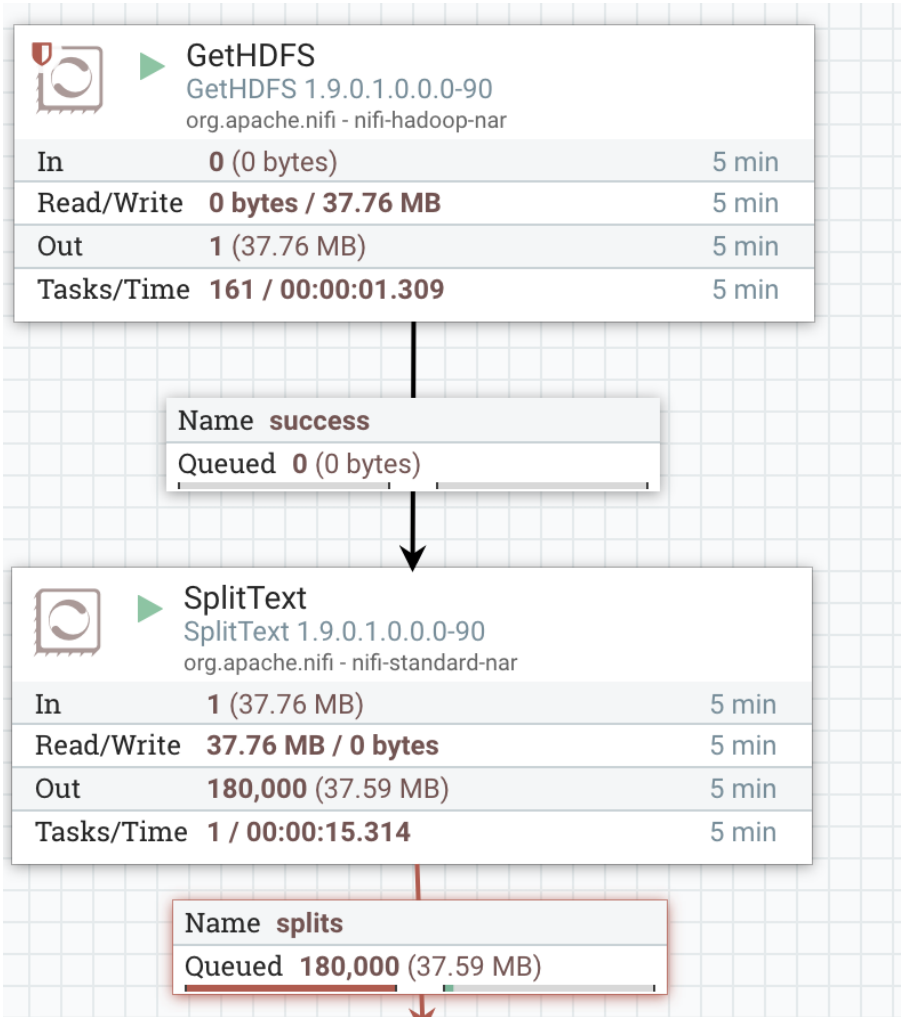
Select the **GetHDFS**, right click and Start



Refresh and then we should have the file in the queue

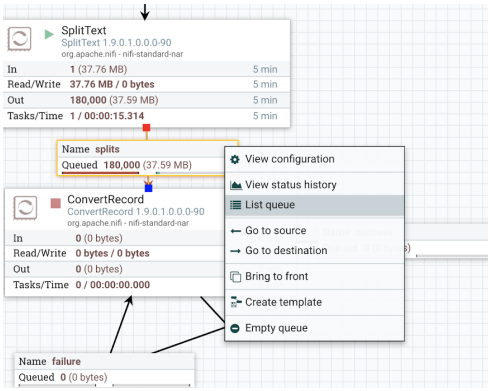


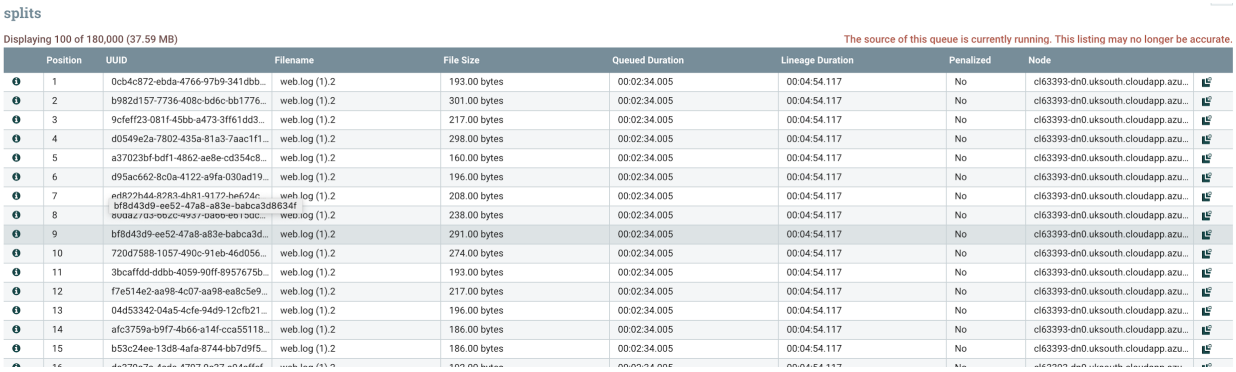
Start the SplitText



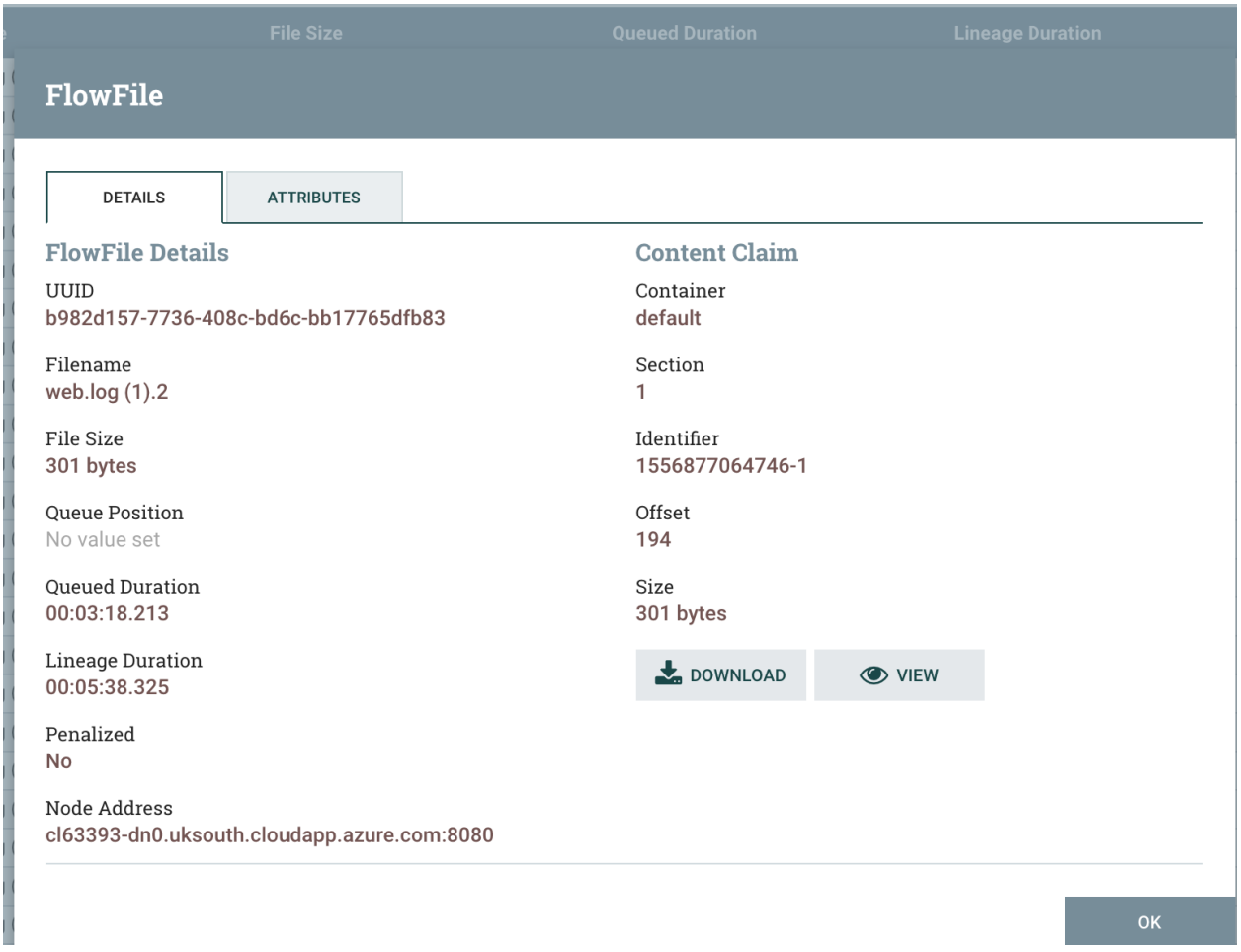
The red line in the queue is normal, it just indicates that the "**back pressure**" control kicked in.

Right click on the queue and **List queue**





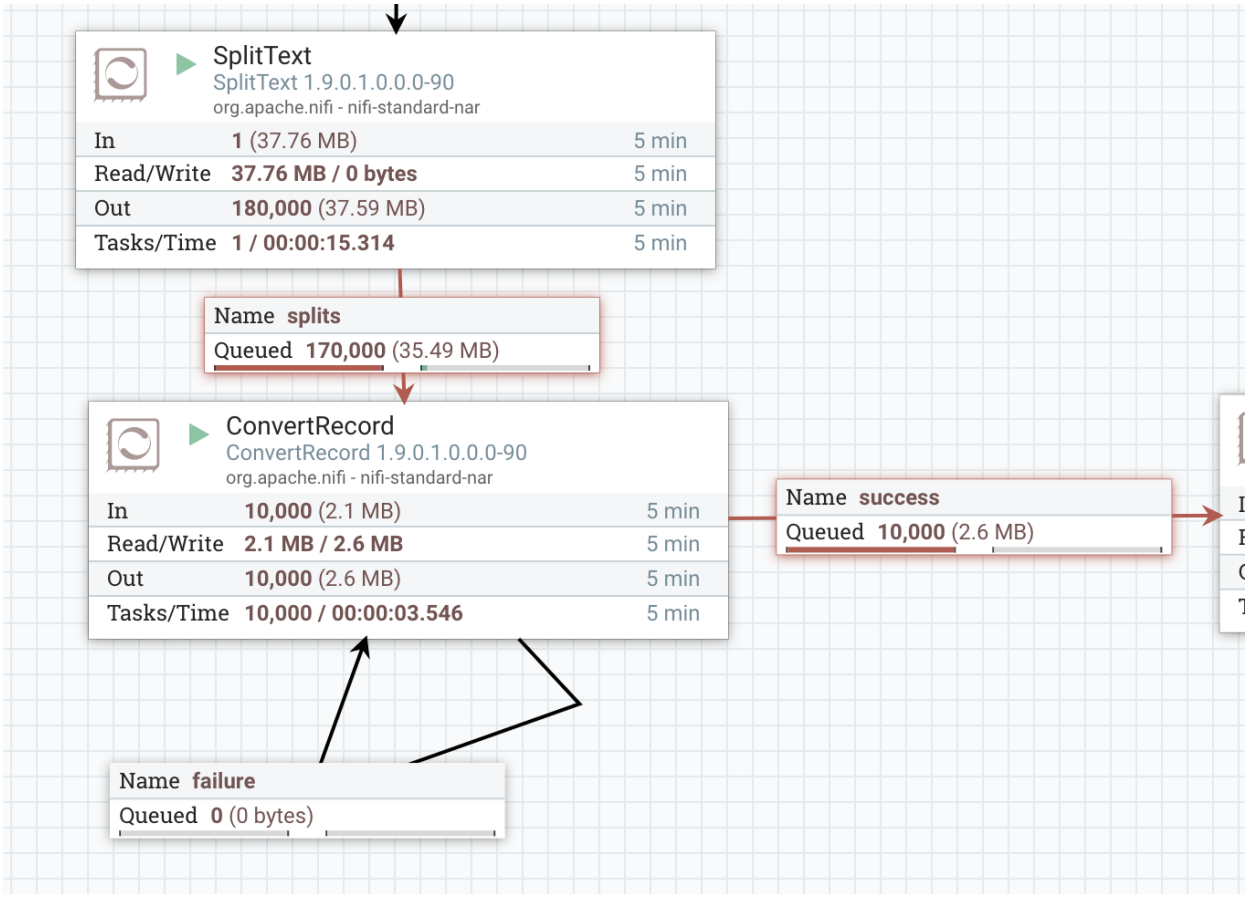
Click on the info icon on an item



and then **VIEW**

this is one line of the web logs file.

Start the **ConvertRecord**

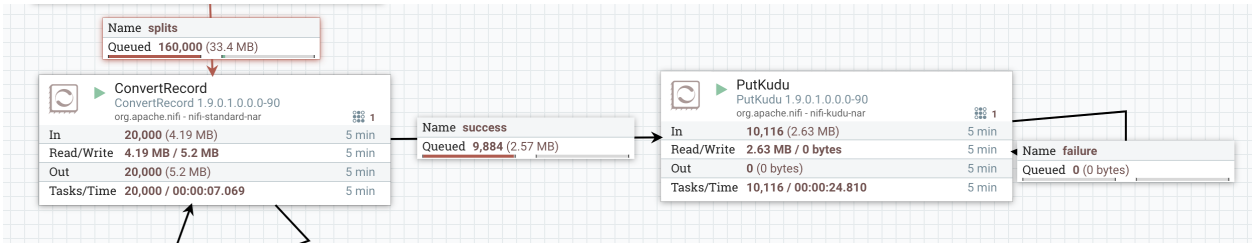


and check on item of the queue that connect the PutKudu

View as: formatted

```
[ {
  "ip" : "79.133.215.123",
  "log_date" : "14/Jun/2014:10:30:13 -0400",
  "log_method" : "GET",
  "url" : "/home",
  "http_version" : "1.1",
  "code1" : "200",
  "code2" : "1671",
  "user_agent" : "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36"
} ]
```

Then run the **PutKudu**



As you can see the queues are getting empty...
In the meantime go to impala and check the weblogs_table.

select count(*) from weblogs_table;

```
16
17 select count(*) from weblogs_table;
```

```
Query f14cc393e9a1653b:5f33bfed00000000 100% Complete (16 out of 16)   f14cc393e9a1653b:5f33bfed00000000
```

Query History | Saved Queries | Results (1)

	count(*)
1	59002

select * from weblogs_table limit 10;

```
18
19 select * from weblogs_table limit 10;
```

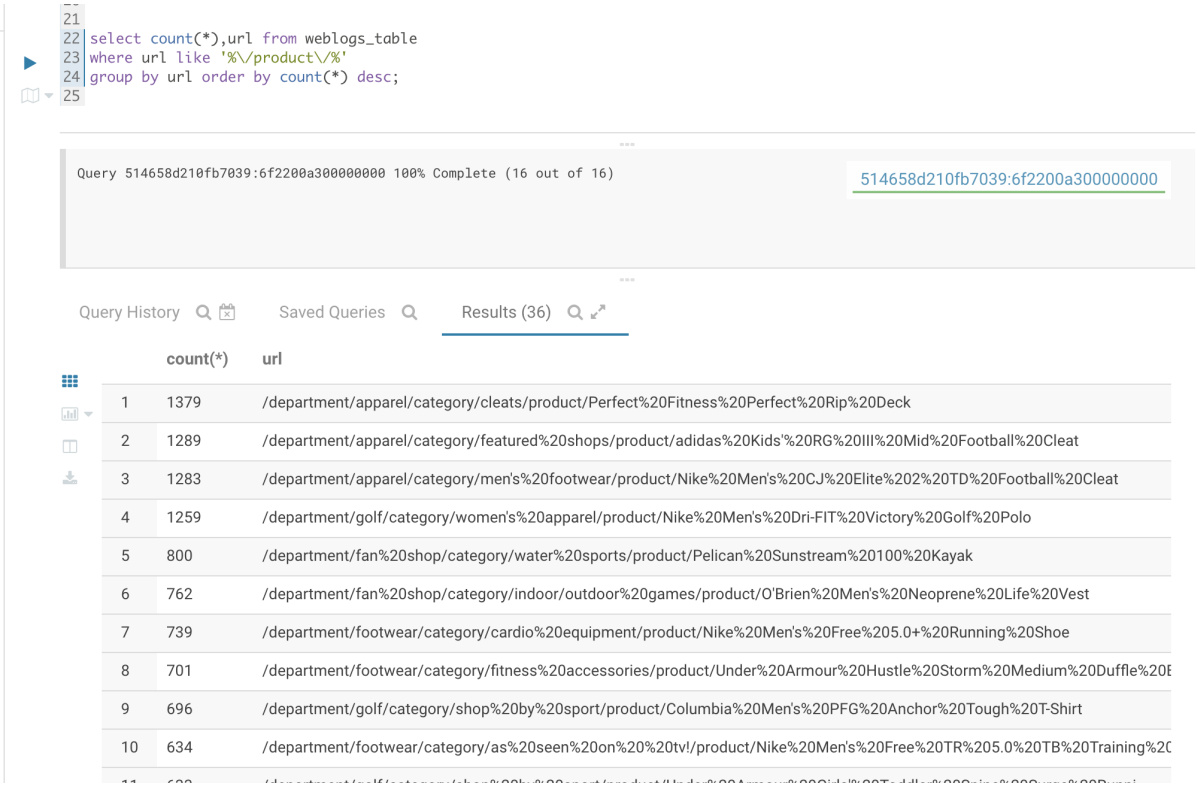
```
Query 80436b72fbfc322c:1c9ab9d700000000: 6% Complete (1 out of 16)   80436b72fbfc322c:1c9ab9d700000000
```

Query History | Saved Queries | Results (10)

	ip	log_date	log_method	url
1	1.111.125.242	14/Jun/2014:22:15:24 -0400	GET	/home
2	1.111.125.242	14/Jun/2014:22:15:49 -0400	GET	/department/footwear
3	1.111.125.242	14/Jun/2014:22:16:12 -0400	GET	/department/fan%20shop
4	1.111.125.242	14/Jun/2014:22:17:46 -0400	GET	/department/footwear
5	1.111.125.242	14/Jun/2014:22:18:02 -0400	GET	/department/footwear/category/cardio%20equipment/product/Nike%20Men's%
6	1.111.125.242	14/Jun/2014:22:19:02 -0400	GET	/department/golf/category/girls'%20apparel
7	1.111.125.242	14/Jun/2014:22:19:36 -0400	GET	/department/fitness
8	1.111.125.242	14/Jun/2014:22:19:43 -0400	GET	/home
9	1.18.156.107	14/Jun/2014:21:51:35 -0400	GET	/department/fitness/category/lacrosse

Try some more queries:

select count(*),url from weblogs_table
where url like '%\/product\/%'
group by url order by count(*) desc;



```
21
22 select count(*),url from weblogs_table
23 where url like '%\/product\/%'
24 group by url order by count(*) desc;
25
```

Query 514658d210fb7039:6f2200a300000000 100% Complete (16 out of 16)

514658d210fb7039:6f2200a300000000

Query History | Saved Queries | Results (36)

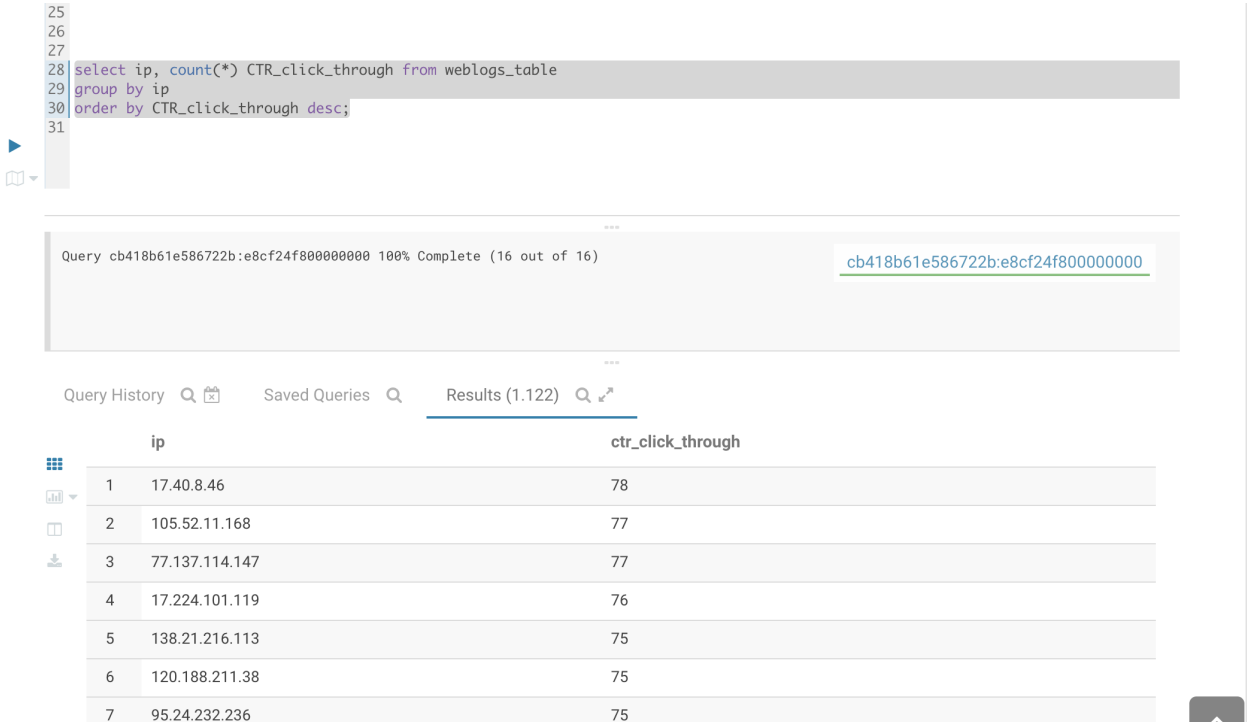
	count(*)	url
1	1379	/department/apparel/category/cleats/product/Perfect%20Fitness%20Perfect%20Rip%20Deck
2	1289	/department/apparel/category/featured%20shops/product/adidas%20Kids'%20RG%20III%20Mid%20Football%20Cleat
3	1283	/department/apparel/category/men's%20footwear/product/Nike%20Men's%20CJ%20Elite%202%20TD%20Football%20Cleat
4	1259	/department/golf/category/women's%20apparel/product/Nike%20Men's%20Dri-FIT%20Victory%20Golf%20Polo
5	800	/department/fan%20shop/category/water%20sports/product/Pelican%20Sunstream%20100%20Kayak
6	762	/department/fan%20shop/category/indoor/outdoor%20games/product/O'Brien%20Men's%20Neoprene%20Life%20Vest
7	739	/department/footwear/category/cardio%20equipment/product/Nike%20Men's%20Free%205.0+%20Running%20Shoe
8	701	/department/footwear/category/fitness%20accessories/product/Under%20Armour%20Hustle%20Storm%20Medium%20Duffle%20E
9	696	/department/golf/category/shop%20by%20sport/product/Columbia%20Men's%20PFG%20Anchor%20Tough%20T-Shirt
10	634	/department/footwear/category/as%20seen%20on%20%20tv!/product/Nike%20Men's%20Free%20TR%205.0%20TB%20Training%20

At the following link you can download the already built flow:
https://cloudera.box.com/s/14hwbdlhqdb8j8cv33dwjnmess9qvicz

Now we will do further analysis – lets try and find out which customer and which IP address had the most click throughs.

Can you pick the IP address that have the highest number of Click Through Rates. Try and create a SQL Query to write this

select ip, count(*) CTR_click_through from weblogs_table
group by ip
order by CTR_click_through desc;



```
25
26
27
28 select ip, count(*) CTR_click_through from weblogs_table
29 group by ip
30 order by CTR_click_through desc;
31
```

Query cb418b61e586722b:e8cf24f800000000 100% Complete (16 out of 16)

cb418b61e586722b:e8cf24f800000000

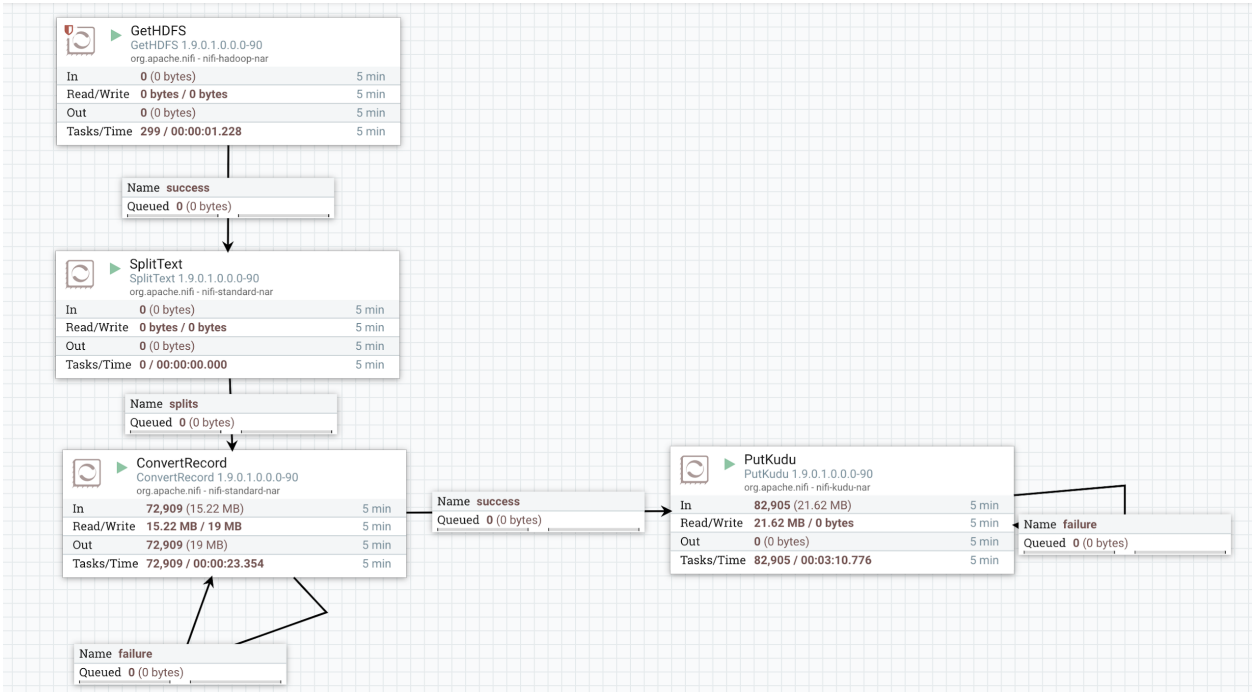
Query History | Saved Queries | Results (1.122)

	ip	ctr_click_through
1	17.40.8.46	78
2	105.52.11.168	77
3	77.137.114.147	77
4	17.224.101.119	76
5	138.21.216.113	75
6	120.188.211.38	75
7	95.24.232.236	75

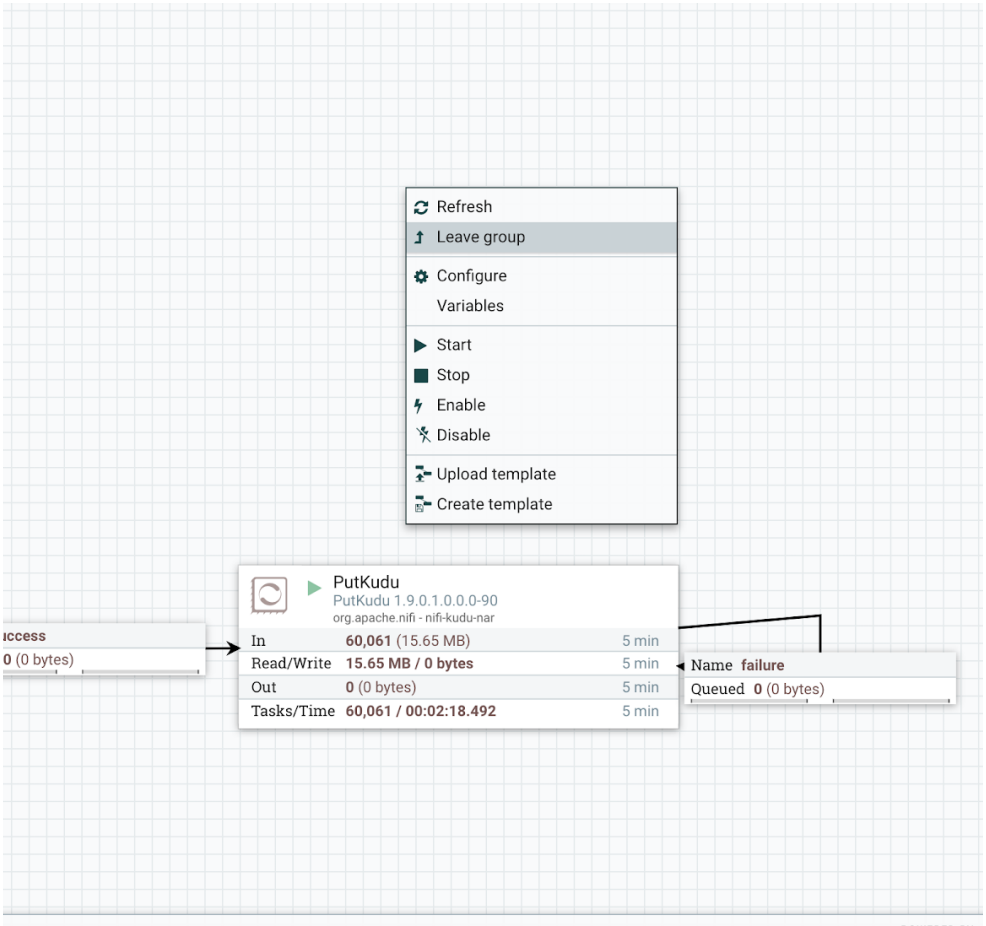
We can do further analysis on where these IP address Geolocation using a pySpark job and finding out which states have the highest number of click throughs on our retail website.

We could even tie IP address to customer based on customer login details and create a Customer360 of our retail customer profile and shopping habits.

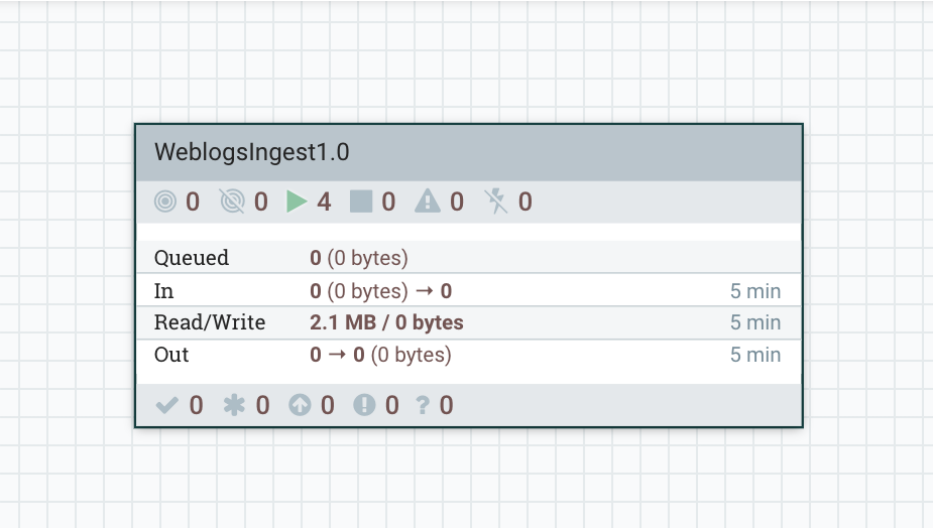
If you go back to NiFi then you should see that all the messages have been consumed:



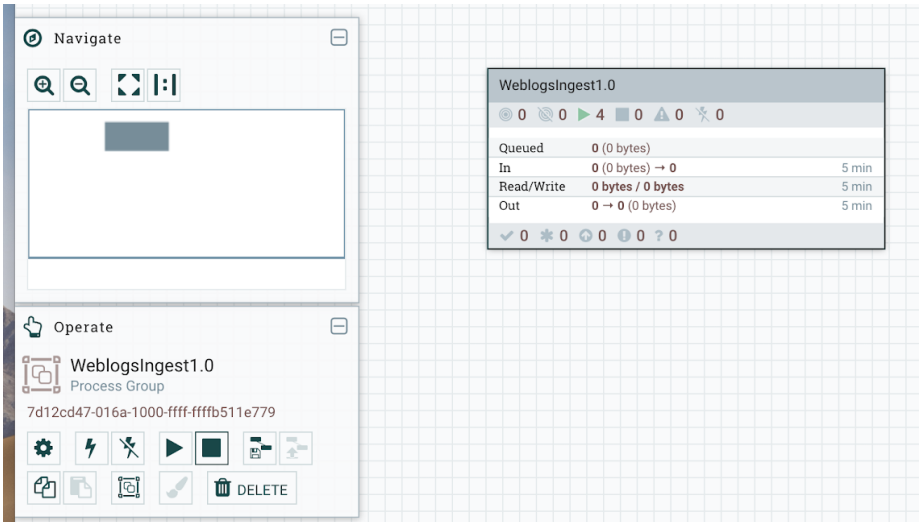
Now you can stop the flow, click on a point in the canvas



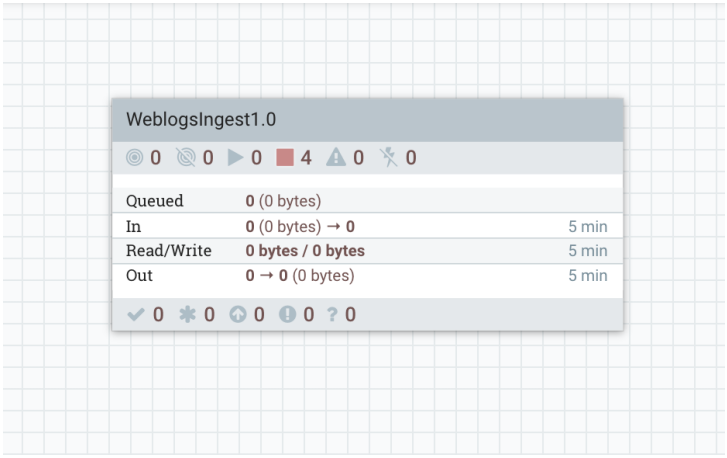
Leave group
then select the WeblogsIngest1.0



and click **Stop** in the Operate box



and our flow is stopped


WeblogsIngest1.0
0 0 0 4 0 0
Queued 0 (0 bytes)
In 0 (0 bytes) → 0 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 → 0 (0 bytes) 5 min
0 0 0 0 0